**Polynomial calculation**

***-Project documentation-***

Student: Cocoş Tudor-Adrian

Group: 30423

**Contents:**

1. Objective of the assignment
2. Problem analysis
   1. Assumptions
   2. Modelling
   3. Scenarios
   4. Use cases
3. Design
   1. Design choices
   2. UML diagram
   3. Class design
   4. Algorithms
   5. User Interface
4. Implementation
5. Testing
6. Results
7. Conclusions
8. Bibliography

**1) Objective of the assignment**

The main objective of this assignment is to propose, design and implement a system for polynomial processing. Consider the polynomials of one variable and integer coefficients.

The secondary objectives are the steps needed to be taken to achieve the main objective. First of all we need to analyse the mathematical problem of computing polynomials and what operations can be performed on them. After that we need to break the elements of the problem into implementable features like classes and methods needed. The next steps include class design, method implementation, User Interface development and testing. Finally, we evaluate the results and think about further improvements.

**2) Problem analysis**

*a) Assumptions*

Given the assignment, we assume that a user must input a polynomial to our application, choose the desired operation and get the right result.

*b) Modelling*

To make the use of the application feel intuitive and closer to the natural way of writing a polynomial, the model chosen is of polynomials of the following kind:

P(x) = 5x^6 – 2.5x^3 + 3.0x^1

As it can be observed, the polynomials represent a sequence of monomials. These are made up of coefficient, variable and the degree. Having to process the polynomials and no to evaluate their values so there is no need for the true value of the variable. It will be considered just a character together with the “to the power” character ‘^’. Even though the assignment talks about integer coefficients, the choice at the lab was to implement them using real values.

*c) Scenarios*

The main scenario and the perfect one, of course, would be a polynomial inputted in order of the degrees of the monomials. But taking in account the possibility for human error the application will be implemented in such a way to take any kind of polynomials because the methods will be written in such a way that if the operation asks for matching degrees we will ensure the correctness of the result.

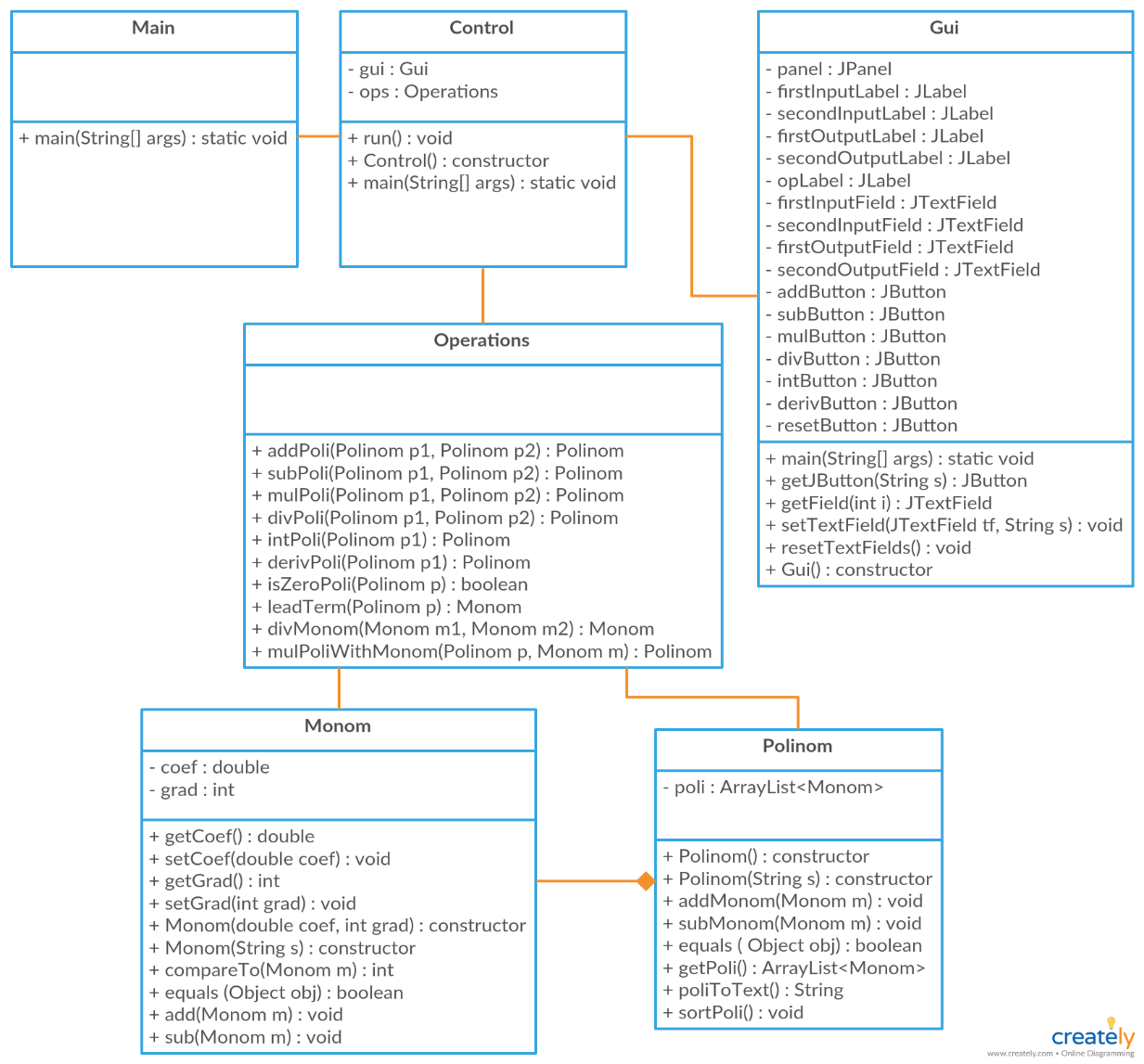
*d) Use cases*

There are a few use cases given the task at hand. If we think about polynomial addition, subtraction, multiplication and division they require two polynomials, while the operations of integration and derivation they only need one polynomial inputted. Of course, if the user forgets to input the second polynomial for the operations of addition or subtraction the result will be exactly the inputted polynomial (in case of the subtraction it could be the inputted polynomial with reversed signs of the monomials). For multiplication nothing will be returned by multiplying with 0 while for division the program will return a message for division by 0.

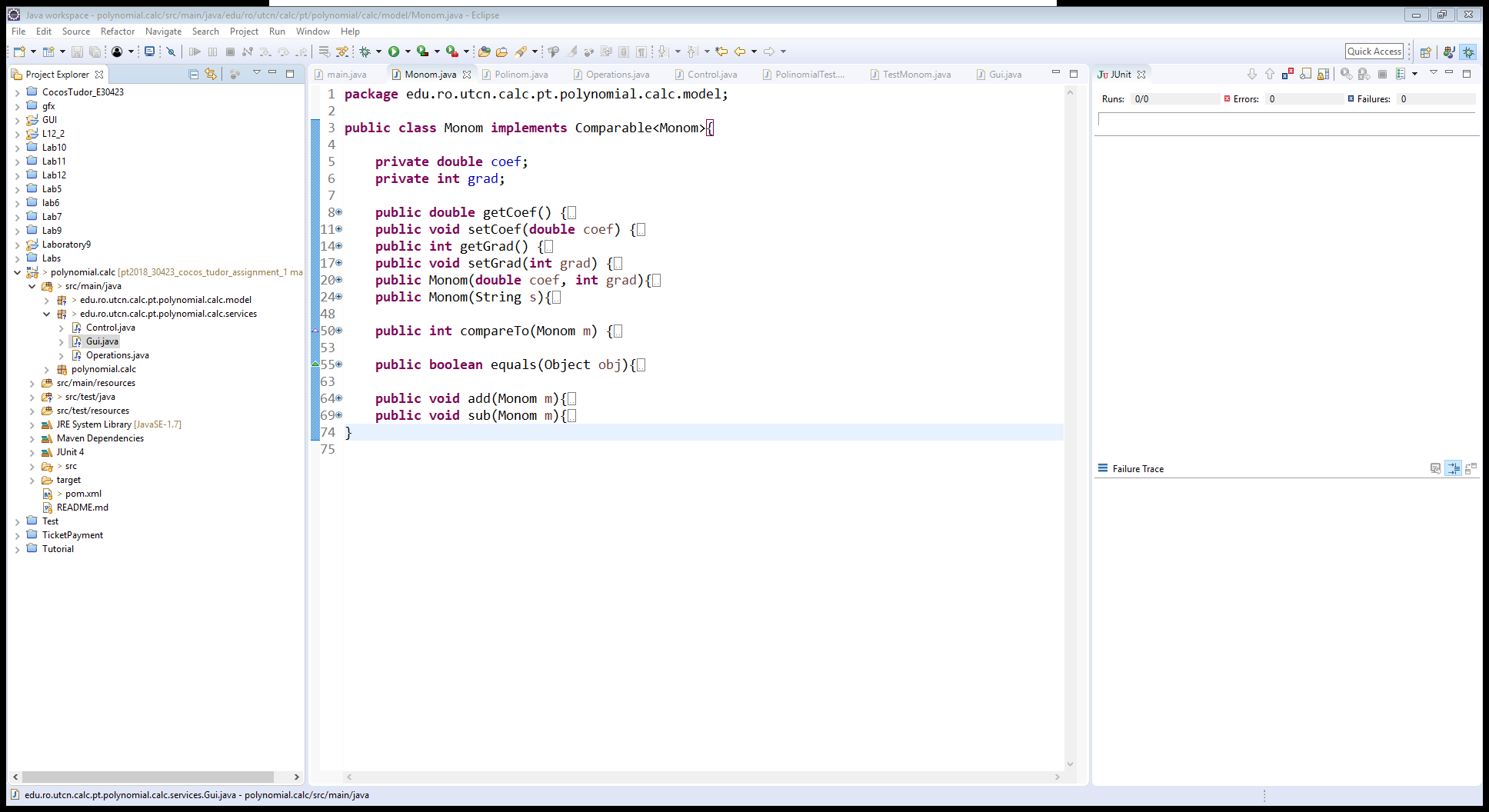
**3) Design**

*a) Design choices*

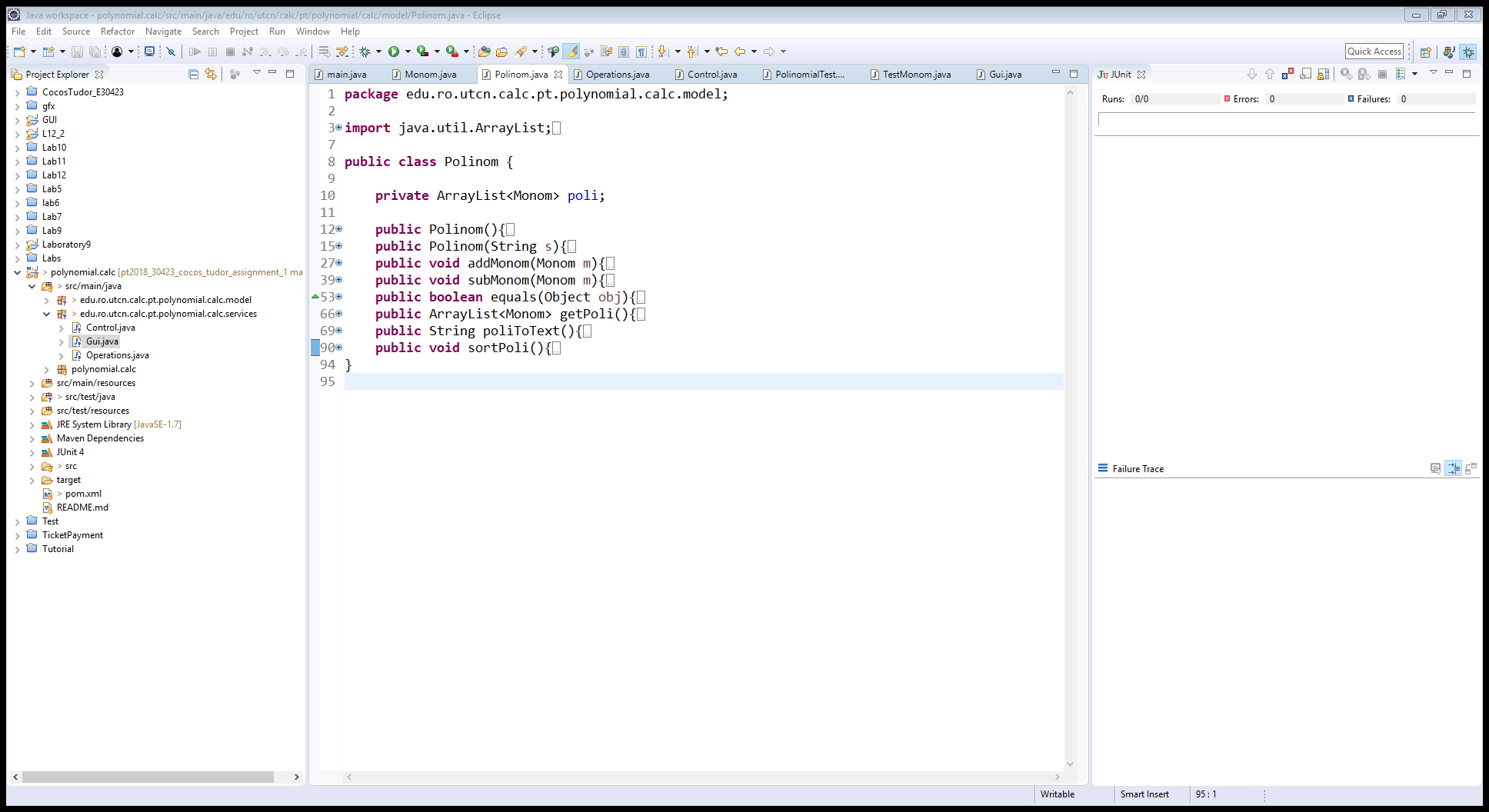
We start class design with the monomial as that is the base of our whole project. A monomial should be defined by its coefficient and degree. Given the choice of real coefficients, values of the double type will be used while the degree will be an integer. Following the monomials are the polynomials which are nothing more but collections of monomials. It results into defining the polynomial into an ArrayList of monomials. This was the base of my thinking as I started to develop the GUI and its Controller afterwards and only on the way to add further methods to the previous classes as the development of the application asked for it. In the middle, between user interaction and the representation of polynomials and monomials in the back, we added an Operations class which executes the chosen mathematical operations.

*b) UML Diagrams*

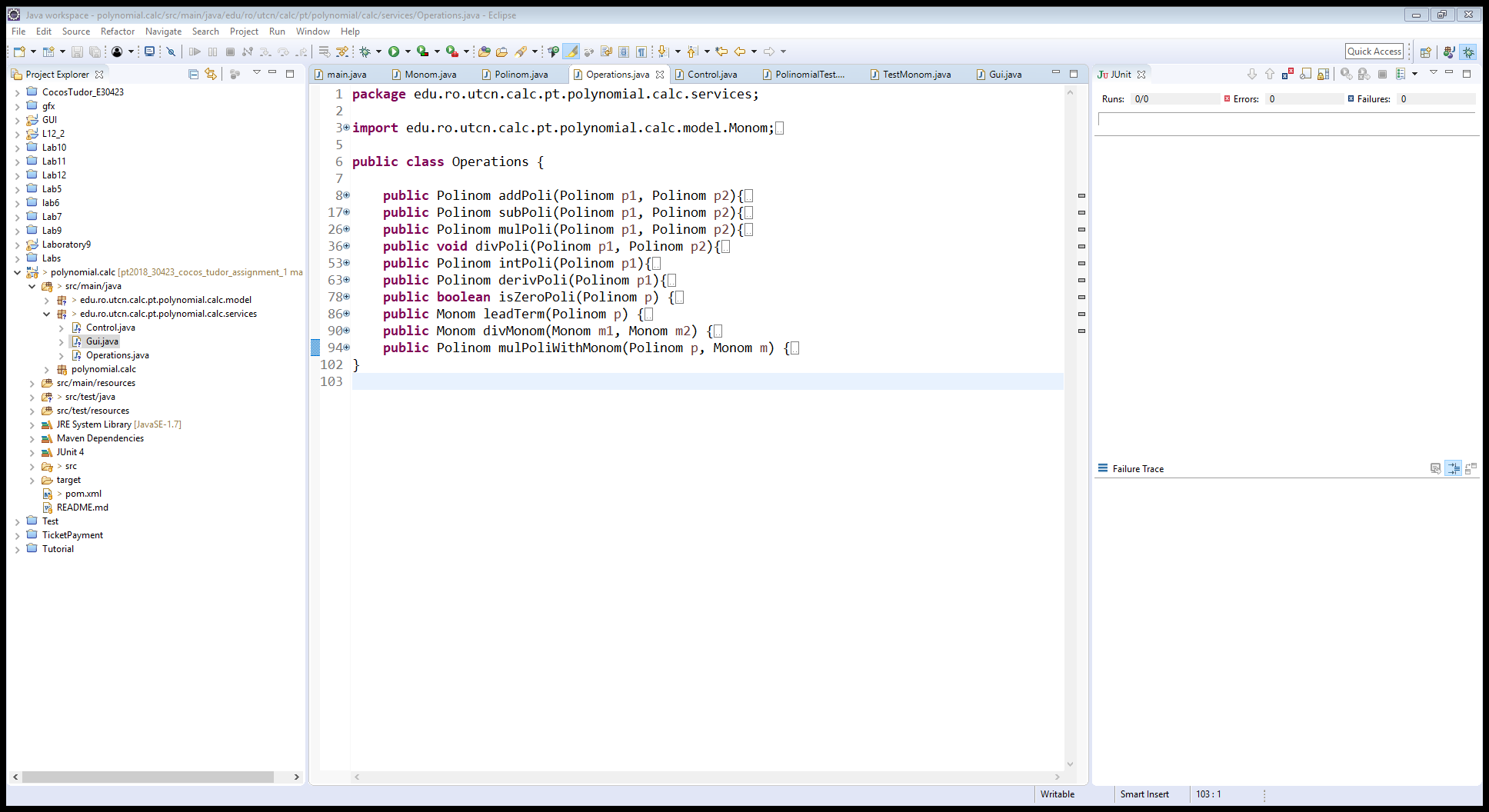
*c) Class design*



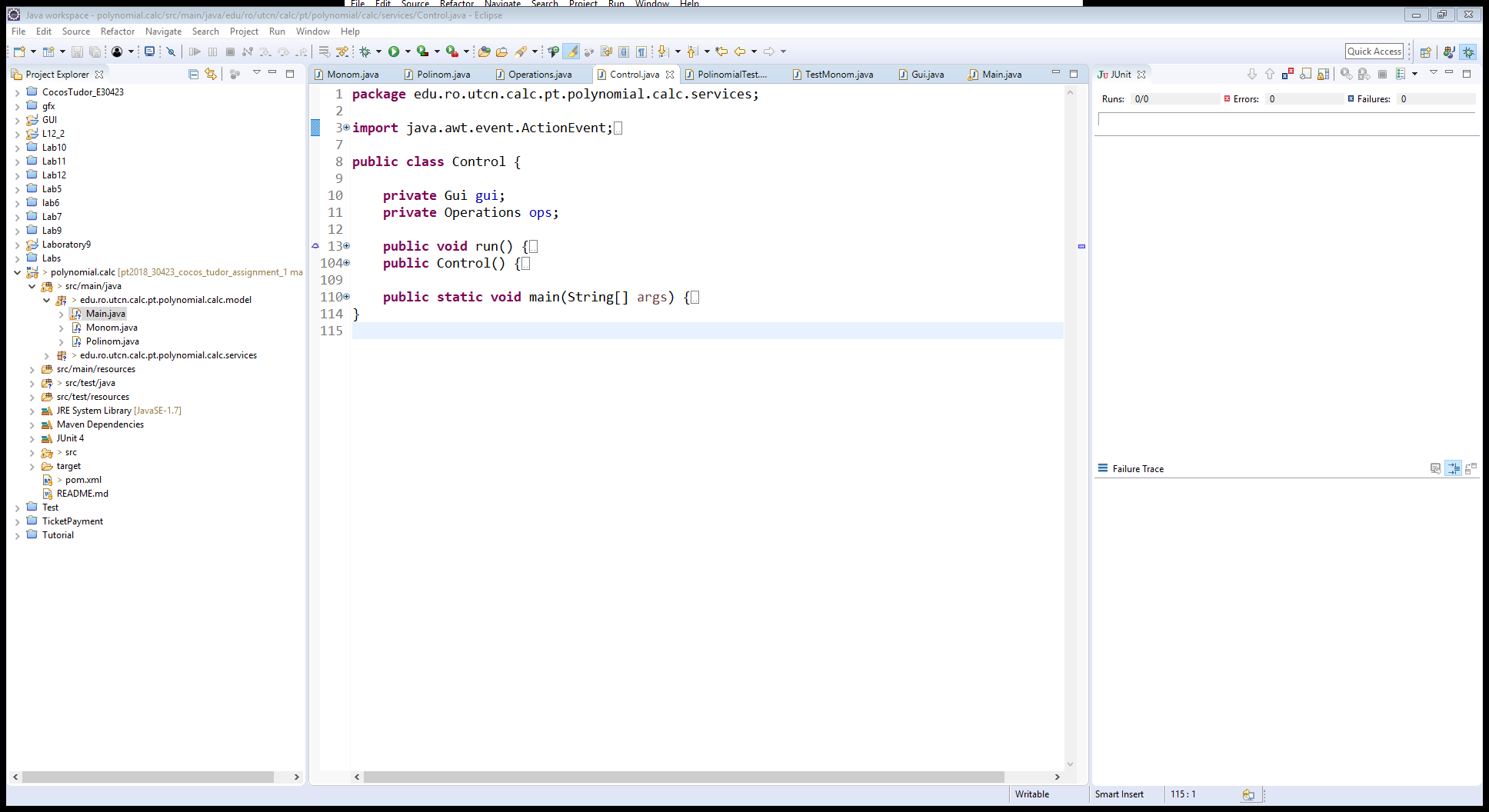
***Monomial class***

******

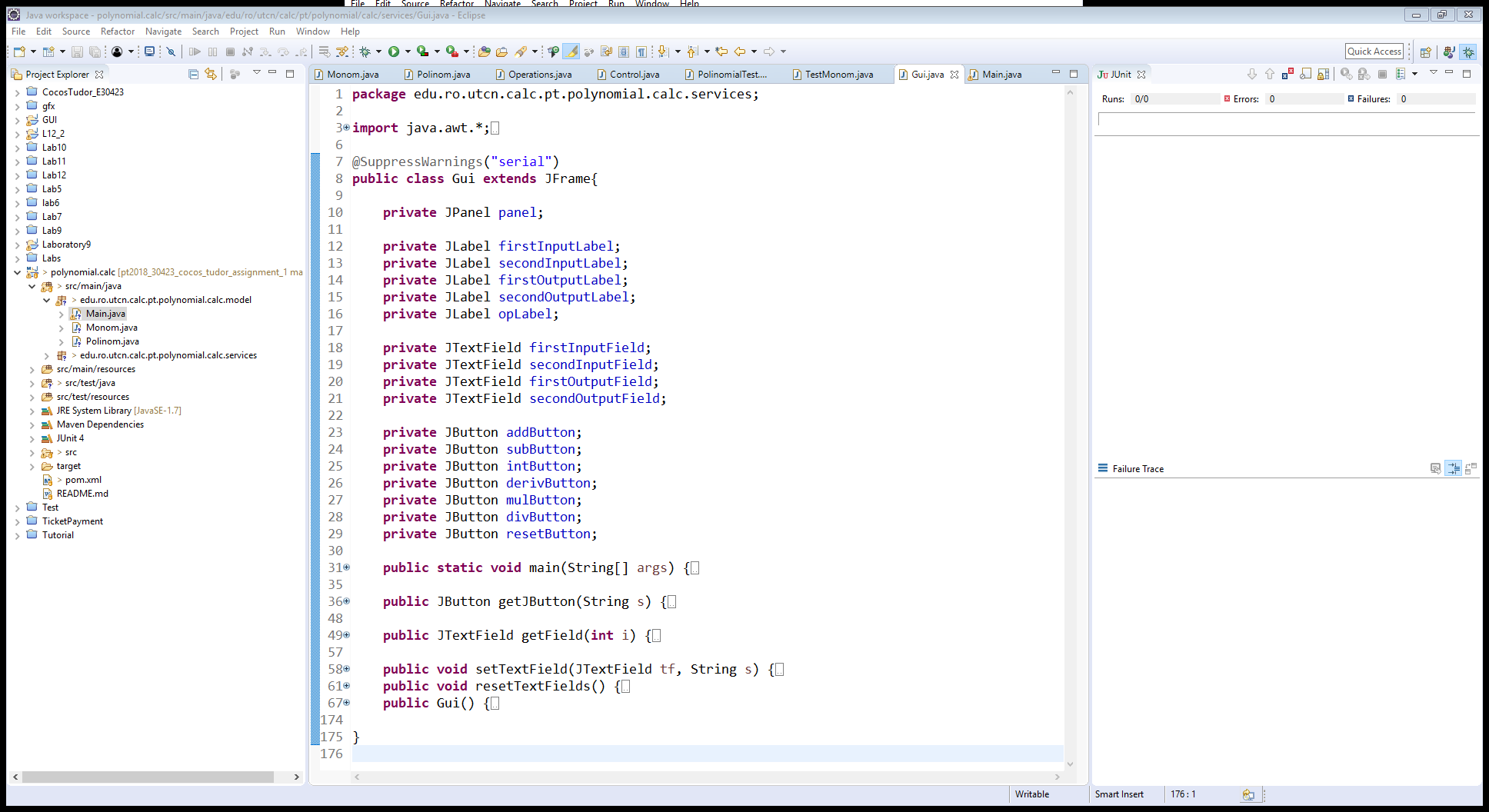
***Polynomial class***

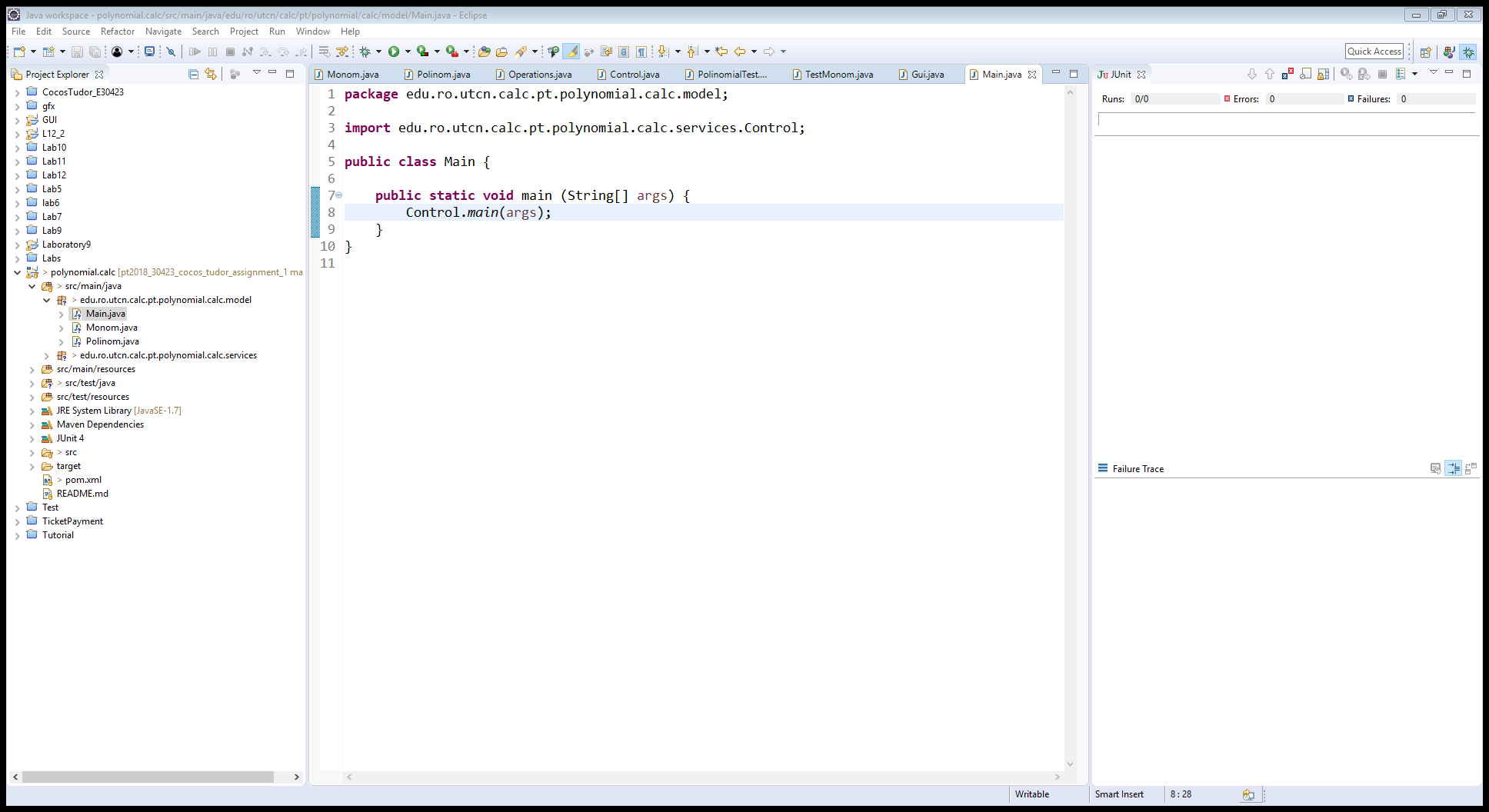
******

***Operations class***

******

***Control class***

******

***Gui class***

***Main class***

*d) Algorithms*

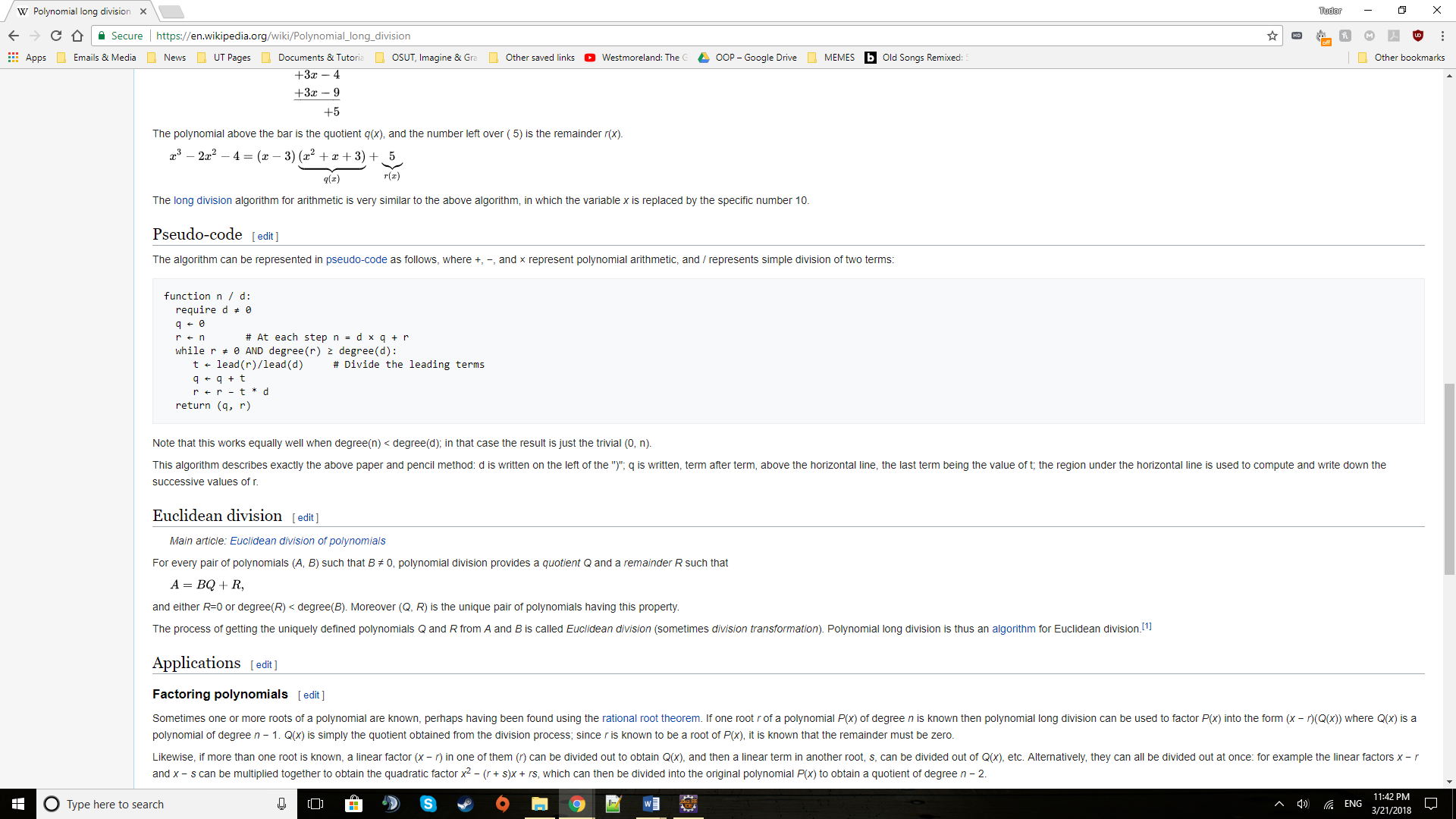
There are six algorithms used to implement the six chosen polynomial operations:

For addition, we take the first inputted polynomial and we add monomials from the second polynomials. If the degrees match => we add the coefficients, else we add the monomial of the second polynomial as a new monomial to the first one with its sign unchanged.

The subtraction is like the addition with the mention: If the degrees match => we subtract the coefficients, else we add the new monomial with the sign changed.

Multiplication is done by multiplying in turn the first polynomial with the monomials of the second one. These result in a series of monomials which are added to a new polynomial that is returned. Using the addition algorithm, we make sure that no terms of equal degree are left because they will be added together.

For division I used the following algorithm which I found on Wikipedia:



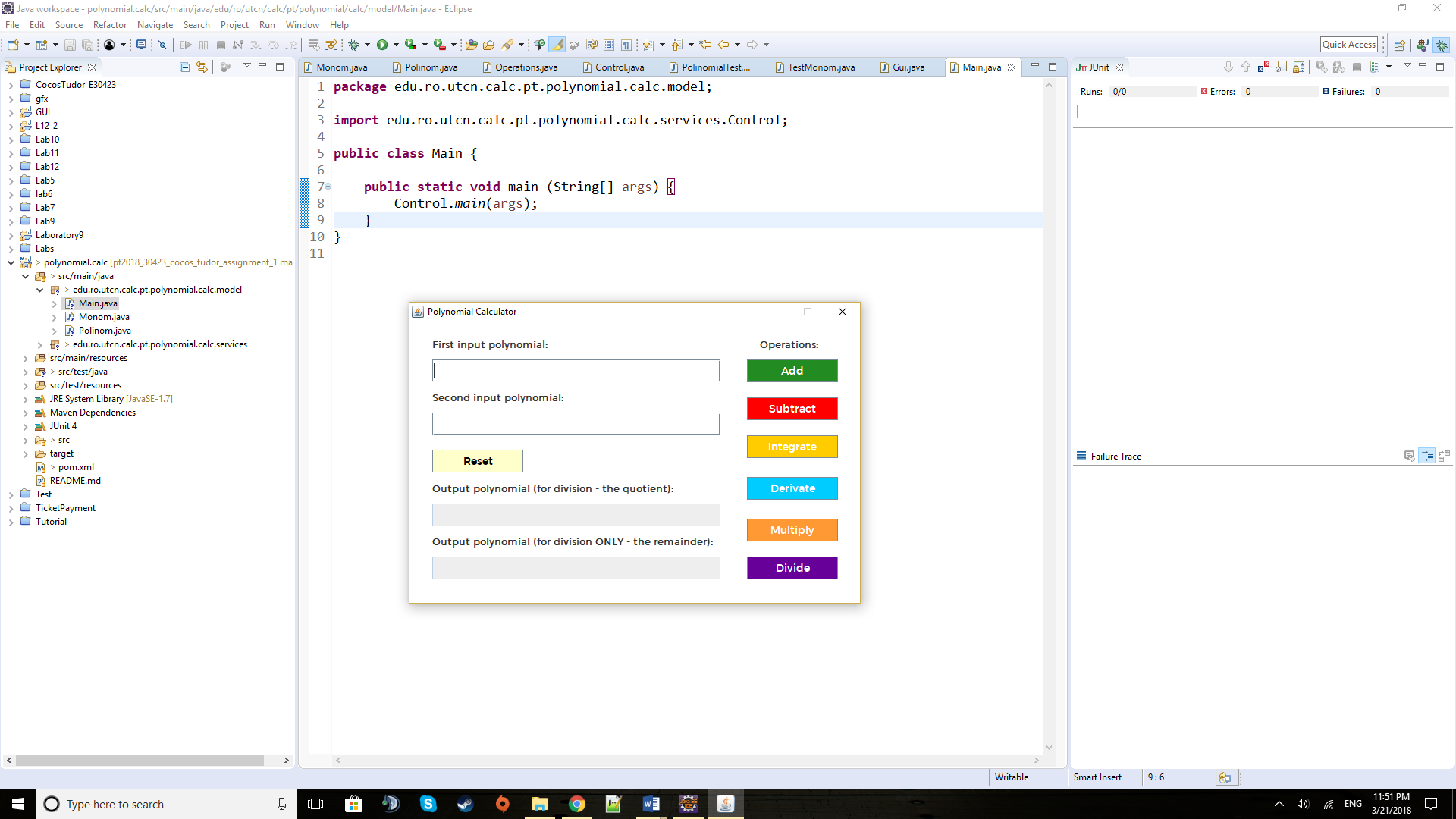
* where n is the first polynomial, d is the second one;
* q is quotient, r is the remainder

Integration is done by taking the monomials of the inputted polynomial one by one and applying the following formula:

coef xdegree => (coef/(degree+1)) x degree+1 and at the end the set of constant values C.

Derivation is the exact opposite operation and is run by the formula:

coef xdegree => (coef\*degree) xdegree-1 with the monomial of degree 0 being removed.

*e) User Interface*

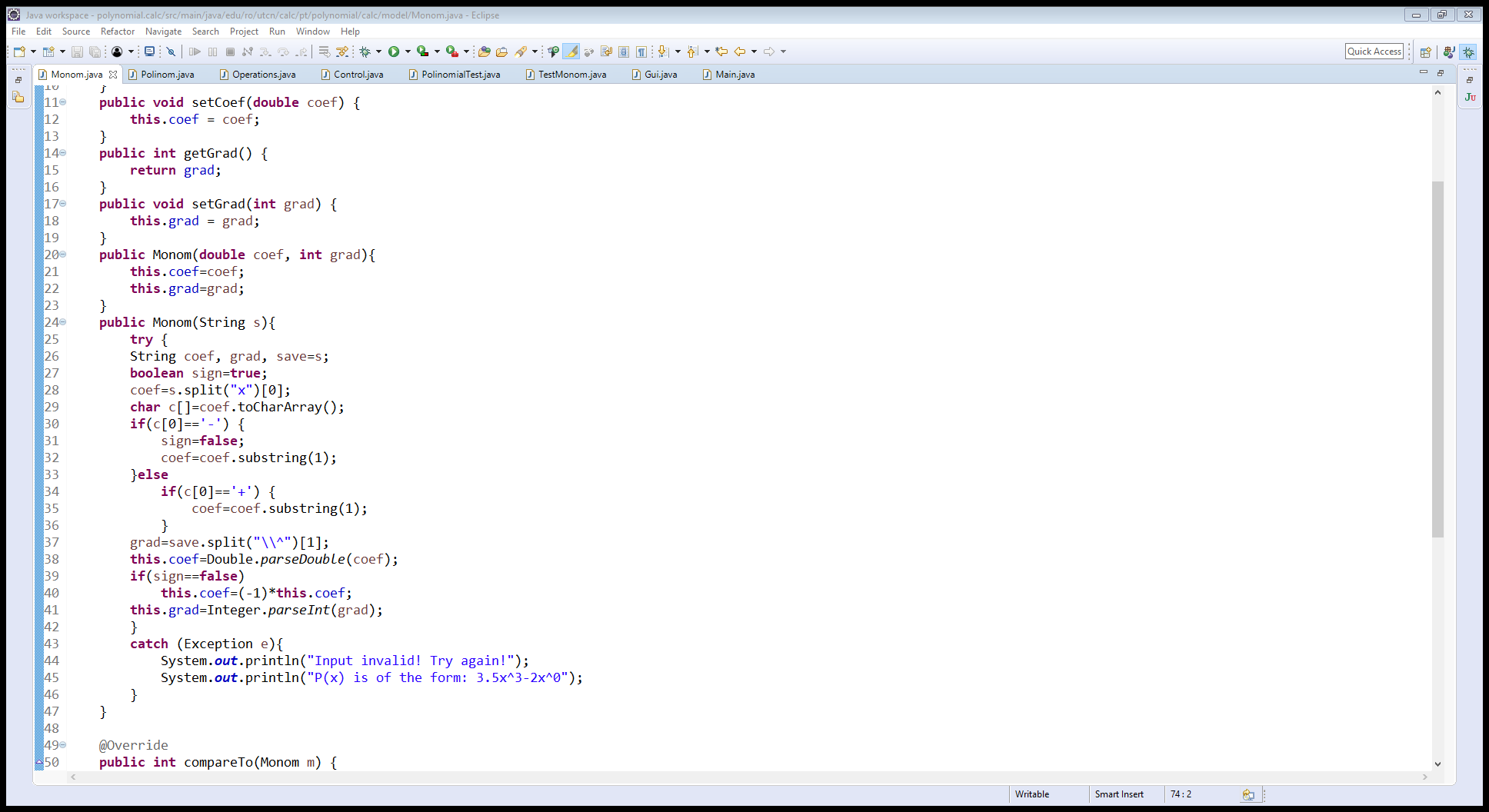
The two white text fields at the top are meant for input while the two greyed ones are used for output. They are greyed out as they are not editable, so the user can’t insert information into them. The light-yellow Reset button is used to reset the text fields and clear them of any inputted or computed polynomials (text, essentially).

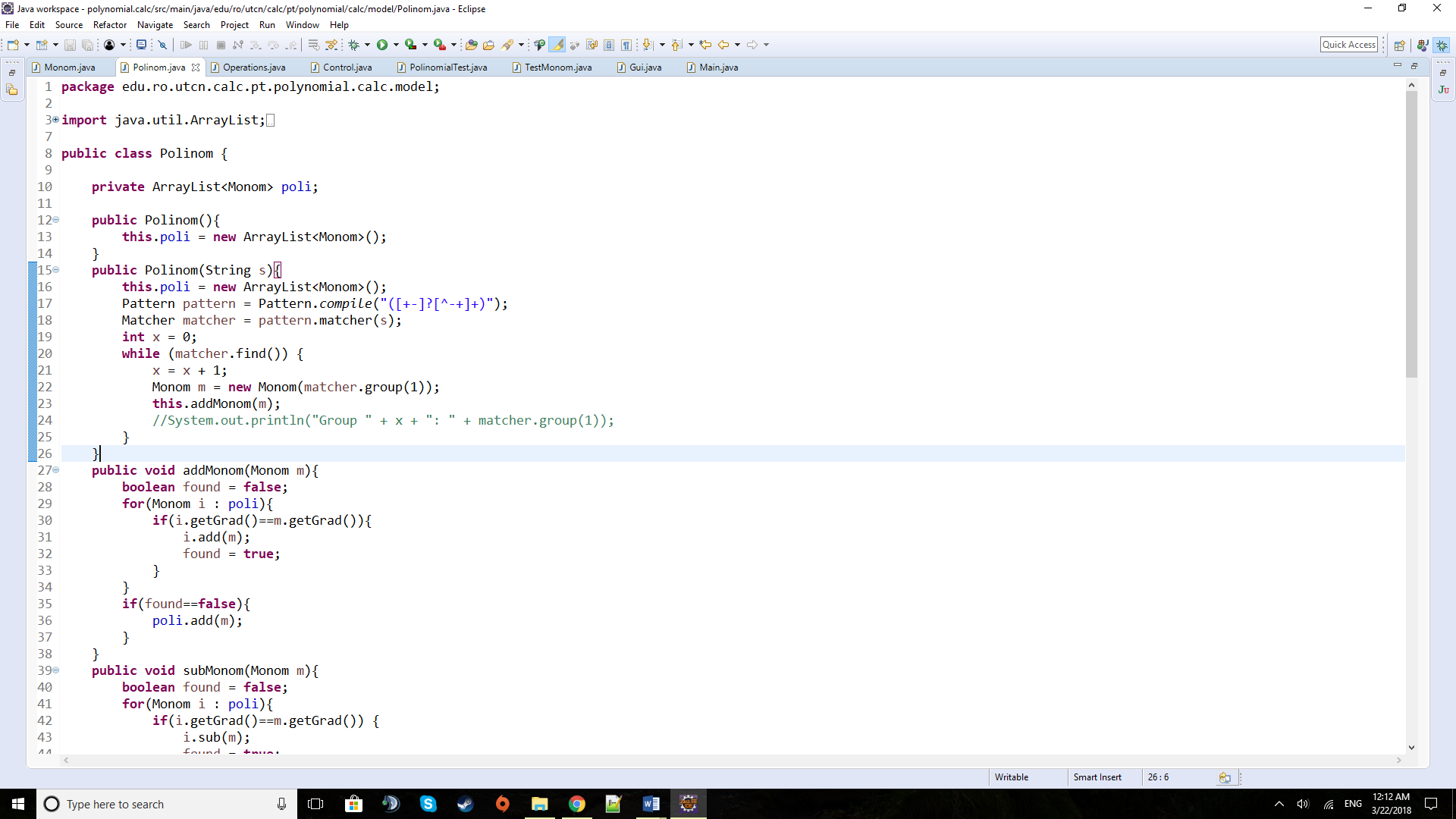
On the right there are the operations buttons which activate the desired operation, and which will make the results appear in the output fields.

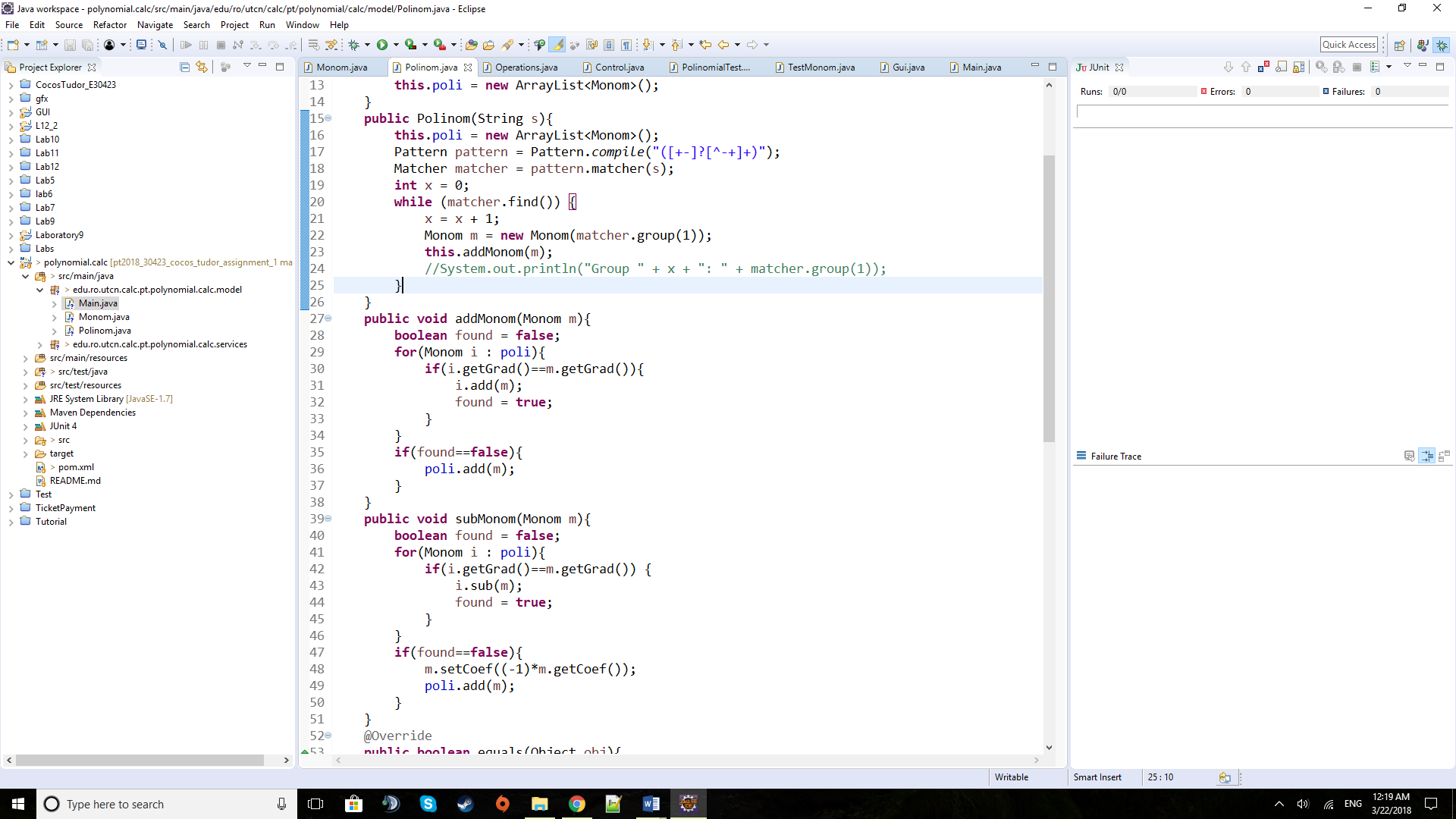
**4) Implementation**

I will describe here the most relevant pieces of code and especially those that are not that common and straight-forward (unlike getters, setters, and some basic constructors).

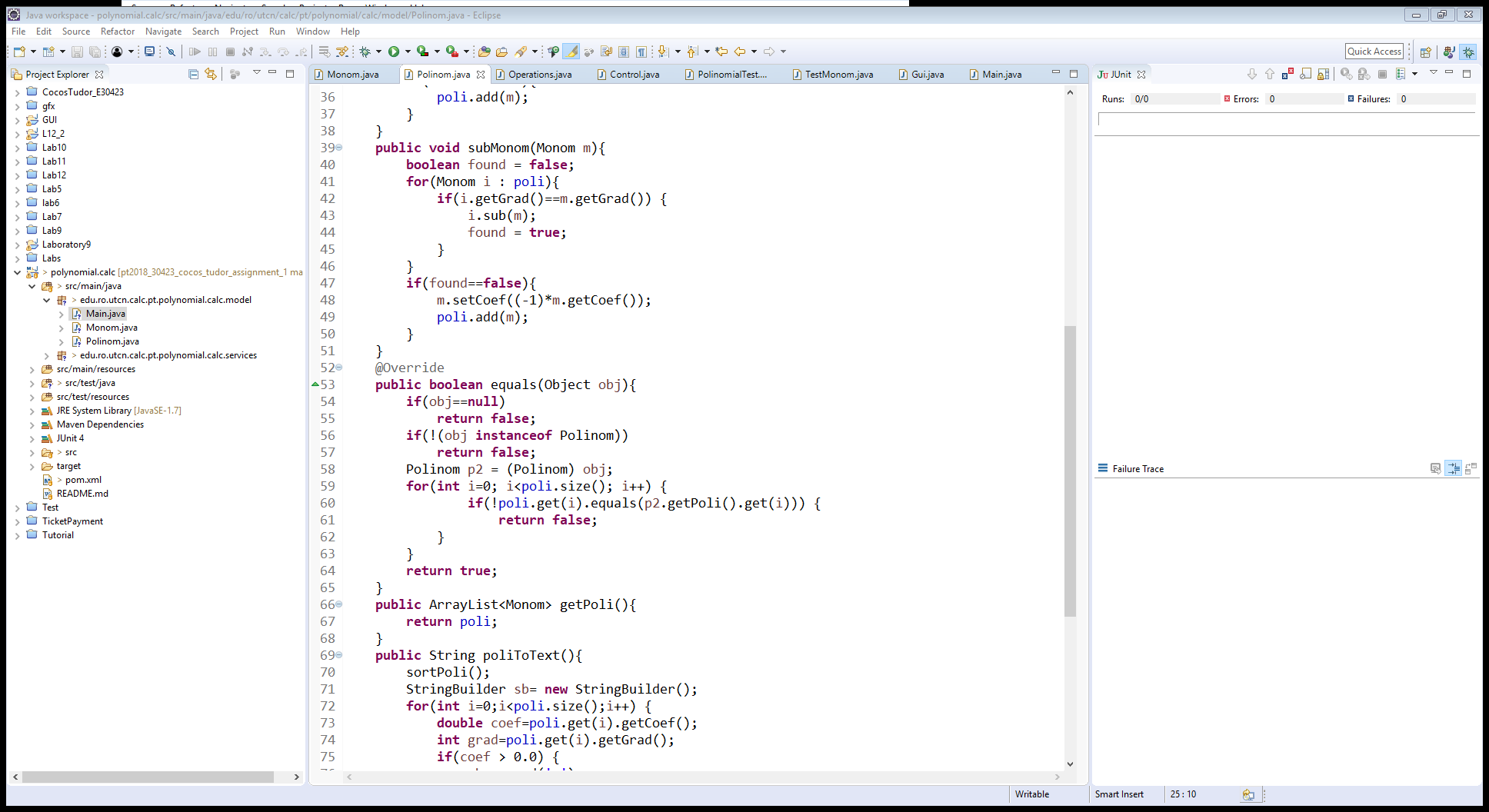
The following code extract is a monomial constructor that takes a string and turns it into a Monom with a real coefficient and integer degree. The inputted string should, ideally, come in the form “sign coef x^grad”. By using the split[0] on ‘x’ we get that is before ‘x’ and that is “sign coef” which we briefly turn into an array of characters to check the first character for sign (which by default is positive -> boolean=true). After that we make another split on a copy of the original large string to get the “grad” after the ‘^’ character. The two obtained strings of coef and grad are turned using parseDouble, and parseInt respectively, into the desired double and integer values that are needed to form the monomial.

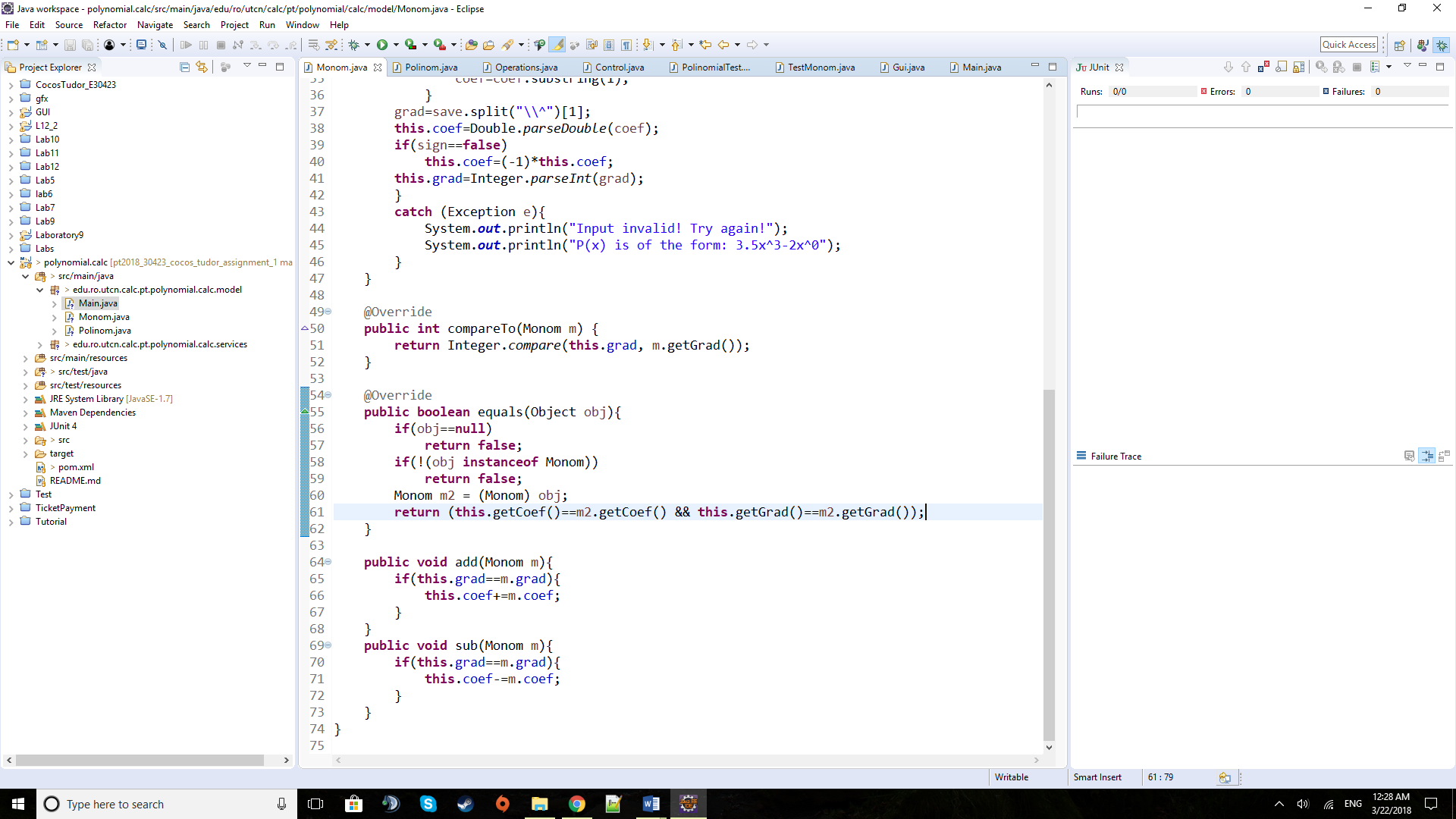


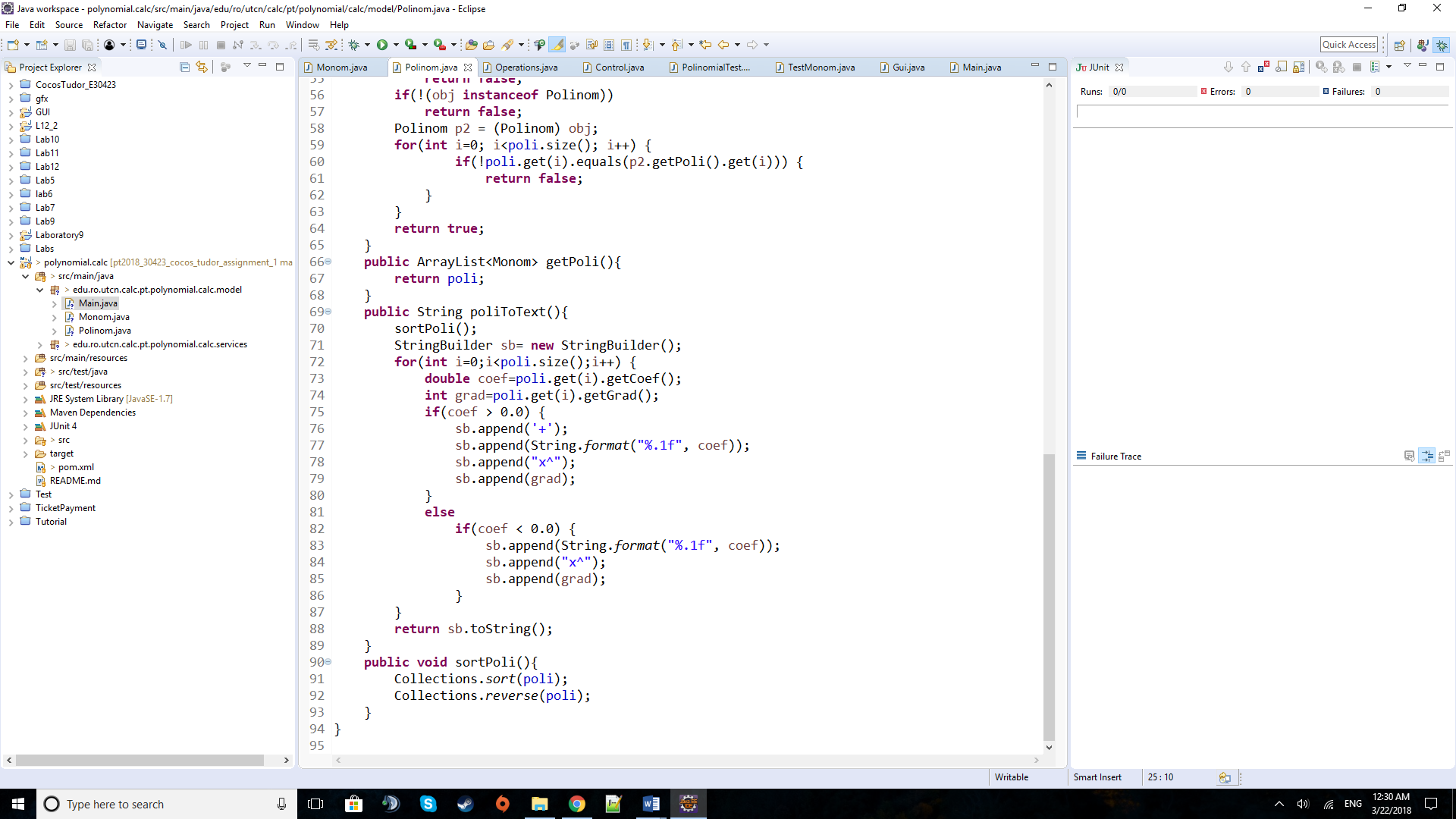
 Getting the string “sign coef x^grad” in the first place is obtained by using the following regex algorithm found online and which splits the larger polynomial text strings into groups determined by the ‘+’ and ‘- ‘ character.



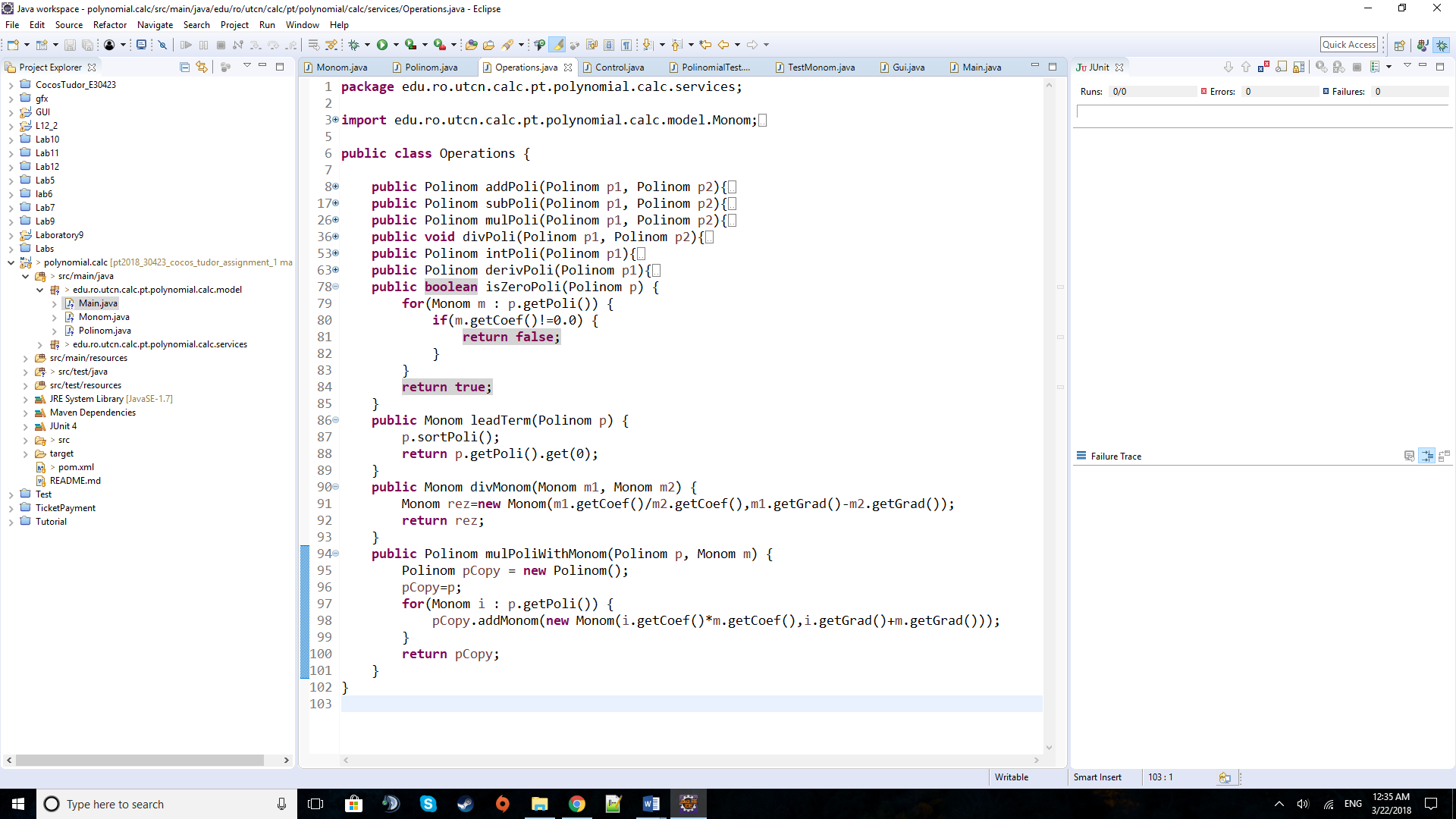
The addMonom method is used to add a new monomial to a polynomial by searching for an existing monomial with matching degree and just adding the coefficients afterwards. Otherwise it will create a new monomial into the polynomial with its given coefficient and degree.



This equals method will prove vital in the testing area when the expected result polynomial will be compared to the computed polynomial. It compares two ordered polynomials monomial by monomial by using an already defined method of monomial equality which checks for equal coefficients to monomials of the same degree:



The poliToText method takes the polynomial and after sorting it in a descending order of the monomial degrees it will construct the output string with the correct signs between monomials and formatted coefficients to only one decimal. It will also only display monomials with non-zero coefficients.

 The methods displayed above are part of the Operations class and are used especially in the division algorithm.

The isZeroPoli checks if the polynomial is a zero one (if it has all coefficients of the monomials equal to zero). It will return false at soon at a non-zero monomial is found.

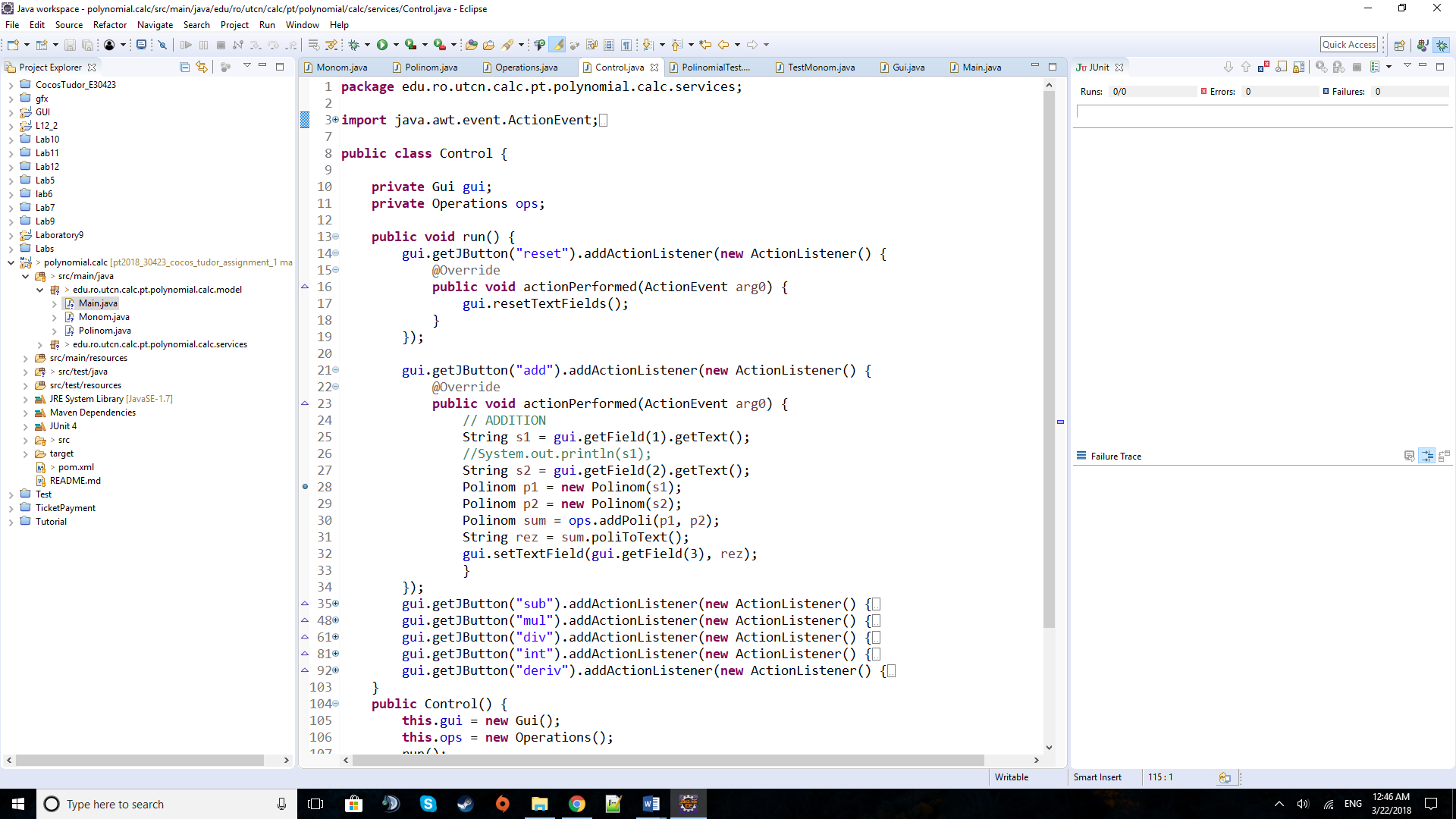
The leadTerm method returns the monomial of the highest degree by sorting the polynomial first into descending order of the monomial degrees and then returning the first element of the array list of monomials.

divMonom is used to divide two monomials by dividing their coefficients and subtracting one’s degree from the other one’s.

The mulPoliWithMonom method multiplies a polynomial with a given monomial. It will take the monomials of the polynomial one by one and multiply it with the monomial given as parameter. These are all added to a copy polynomial that in the end is returned.

The following method is found in the *Control class which is the “brain” behind the GUI. It should be considered part of the UI, but this class only deals with the actions performed by the graphical elements. These graphical elements are described in the Gui class which only deals with the graphical aspects of the application.*

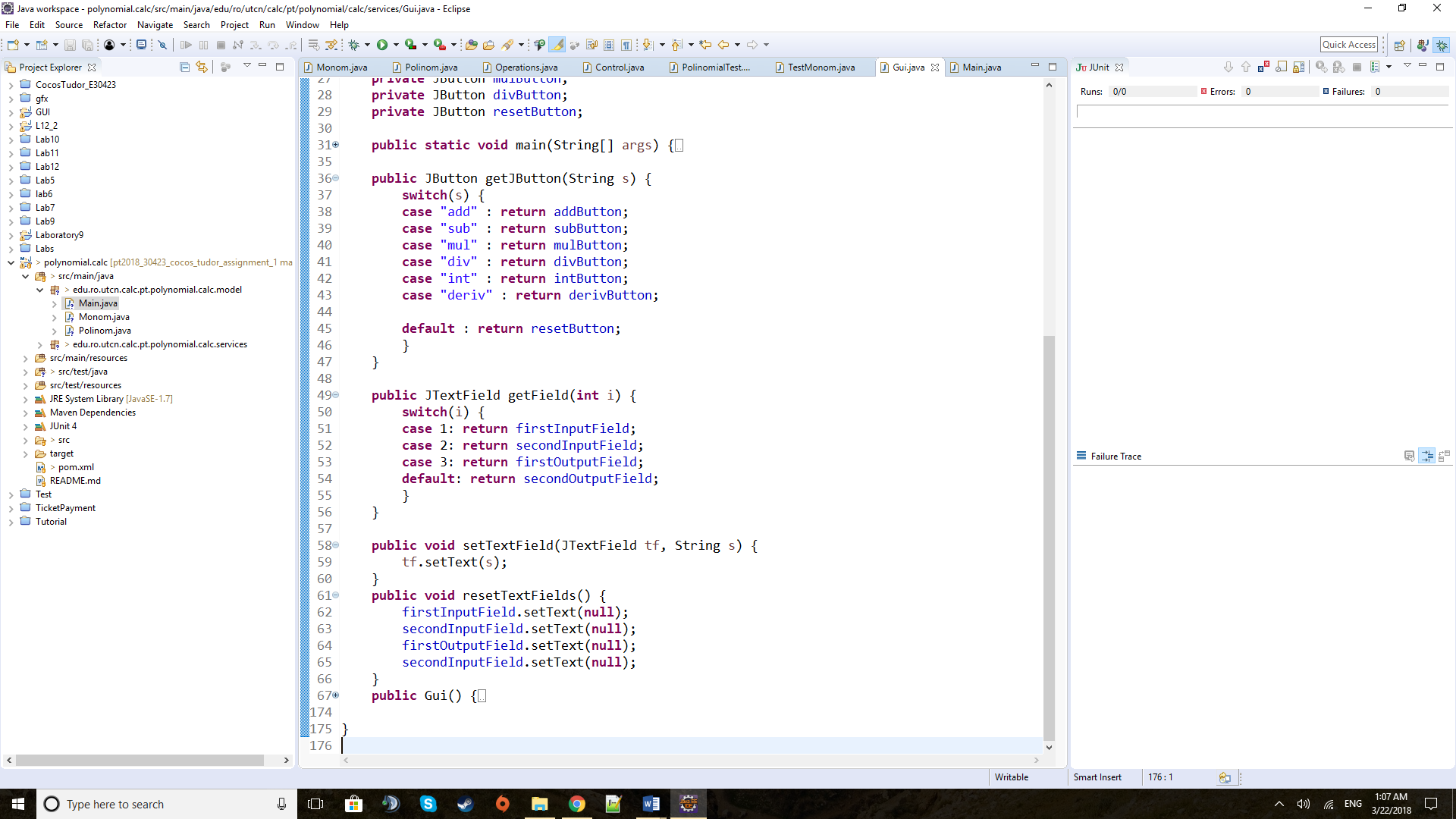
So, the run method contains the Action Listeners of the GUI and are of the following type: the “reset” action for the Reset button and the “perform operation” for each Operation button. These operation methods are very similar, so I would only present only one of them:



The reset action is linked to the Reset button and by pressing it, the resetTextFields (defined in the Gui class) will be called, clearing the fields of texts.

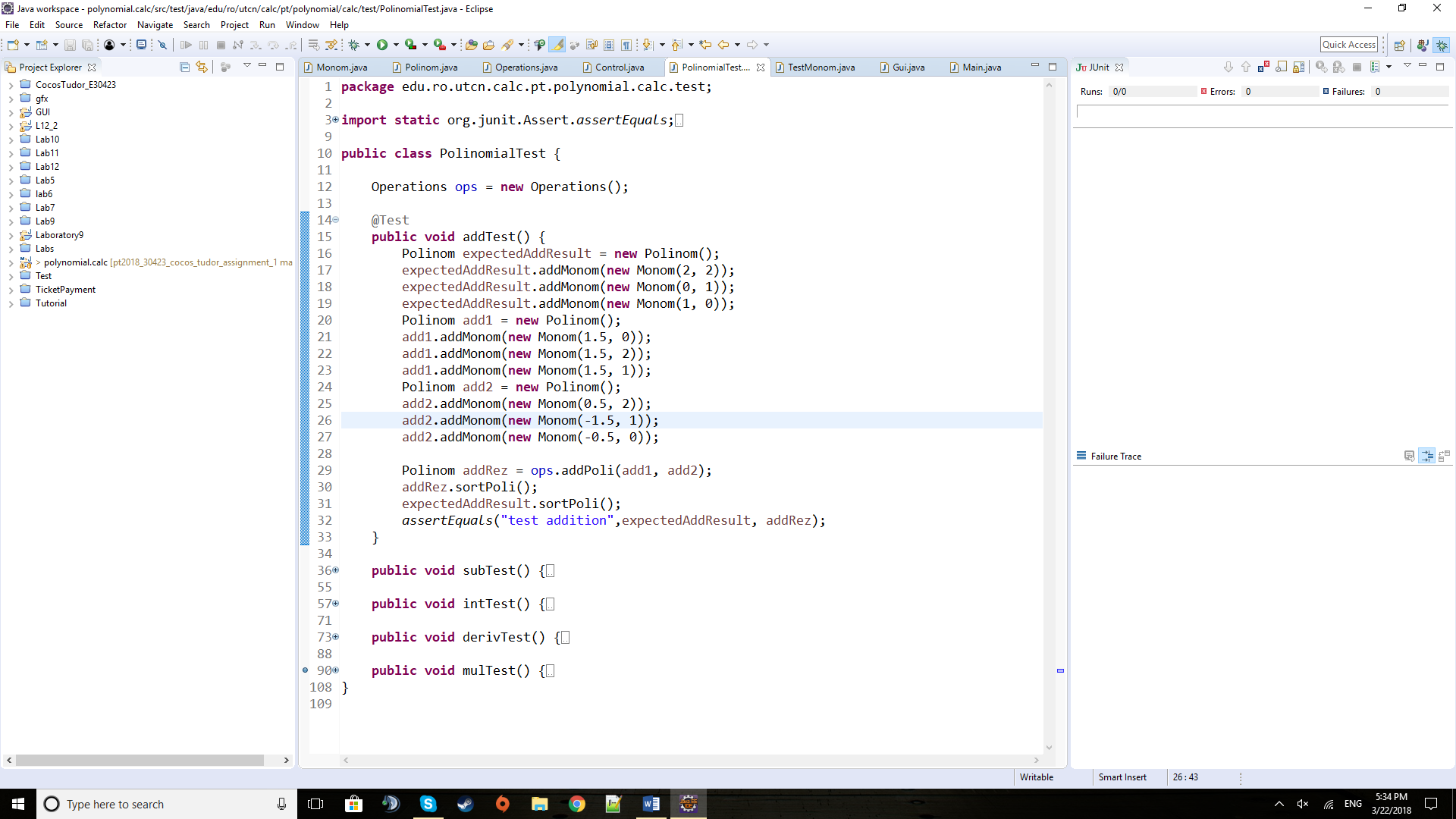
By pressing any of the Operation buttons, one of the following “perform operation” action listeners will be activated, and they will take the strings inputted in the text fields and use the polynomial constructors that have a string as parameter to build the two needed polynomials. After that a new polynomial of the result will be made using the addition of two polynomials. The result is turned back into a string by calling the poliToText method and then it is sent to the desired output text field.

The following methods are part of the Gui class and are some kind of getters and setters (actually one of it is a “resetter”). The getJButton and getField returned the desired JButton, JTextField respectively, by using case instructions. For the one with the buttons a short string that is defining for the operation was chosen to differentiate between them, while for the fields an integer representing their order from top to bottom (1 to 4 ). The setTextField may look like a simple setter for the given string as parameter to the given text field but having been used in a class outside the Gui one it might feel strange, but this is its actual functionality. The last one is a “resetter” as it resets the text fields by setting their text attributes to null.



**5) Testing**

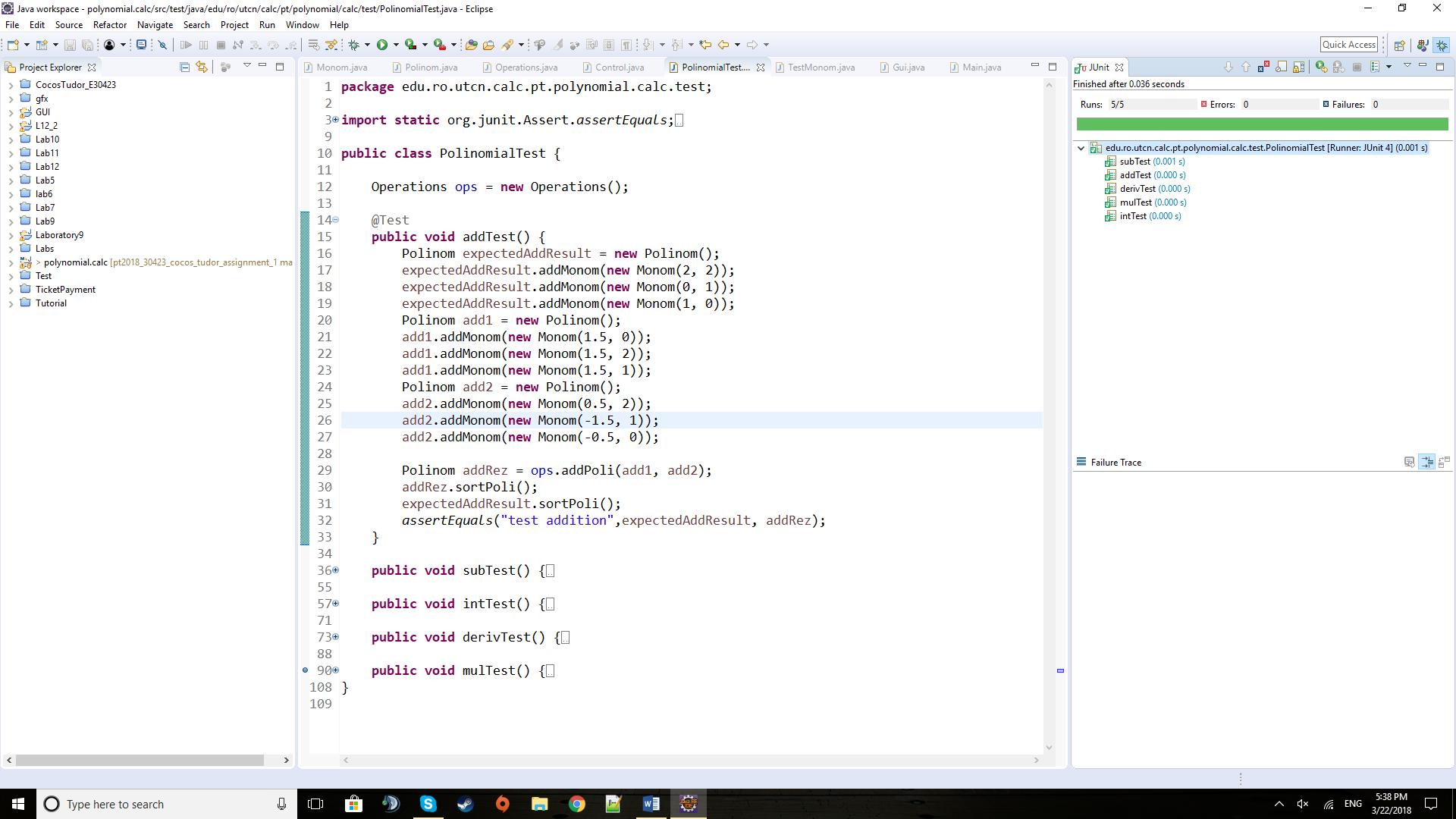
Just implementing an algorithm is not enough so it requires intensive testing to prove its reliability. Besides having inputted a lot of cases during development and afterwards, I have implemented 5 Junit Tests to prove the correctness of my implementation of the 5 operations that are working (unfortunately, I couldn’t make the division work properly). These five operations are: addition, subtraction, multiplication, integration and derivation. For each one of these I have constructed the input polynomial(s) and an expected result polynomial and then, using the equals method implemented in the Polinom class, managed to compare the results of the operations to my expected computations.



Here(above) is how the test class for polynomials looks with its 5 independent test methods that look really similar, with changes only in variables and the number of needed inputs (for integration and derivation).

**6) Results**

The results of running those 5 independent tests came positive, proving the correctness of the implementation.



**7) Conclusions**

During this assignment I have managed to refresh my knowledge of OOP, algorithm design and also mathematics. At the beginning, it didn’t come to me as much of a challenge but during the implementation, when faced with certain obstacles I have seen how much is always to be learned…and I still need to do! Something which might seem small, but I have learned a great deal out of it was the fact that the computer is as smart as it is programmed. This happened during testing when the multiplication test was failing again and again, and I had to debug it. It appeared that 3 \* 1.3 was not equal to 3.9 as the machine was estimating the value from binary to double representation in 3.900000000004, which, of course, could never be equal to the estimated result of 3.9. In the end I regret not managing to finish the division on time.

As for further developments, I would make this into a polynomial calculator that can also take values for the polynomial variables like ‘x’. By including in the monomial also the attribute of the variable, a more advanced application can be developed to compute and process polynomials of multiple variables. For additional operations, I am thinking of root finding while for additional features (especially on the UI side) a history of the recent operations and their operands would be useful.

**8) Bibliography**

For the division algorithm: <https://en.wikipedia.org/wiki/Polynomial_long_division>

For the regex algorithm: <https://stackoverflow.com/questions/36490757/regex-for-polynomial-expression>

For the JTextFields: <https://docs.oracle.com/javase/tutorial/uiswing/components/textfield.html>

For the fonts and colours used: the font setup and the colour palette on the Adobe Illustrator desktop program.

For the UML diagram I used an online application: <https://creately.com/blog/diagrams/uml-diagram-types-examples/>