1. Description of the assignment

We are required to design and implement a polynomial calculator with a dedicated graphical interface through which the user can enter polynomials, select the operation to be performed (i.e. addition, subtraction, multiplication, division, derivative, integration) and display the result.

For reaching that, we need to firstly do the following:

* Create two separate classes, one for Monomials and one for Polynomials, along with all their methods (I am going to touch on this in the 4th field of this documentation)
* Implement the pattern that the inputs should follow using Regex (further explained in chapter 3)
* Create the Graphical User Interface that will help the users to interact with the program using Java SWING (further explained in chapters 3 & 4)
* Use JUNIT for testing if the program works accordingly (further explained in chapter 3)

1. Problem analysis, problem modeling, scenarios, utilization cases

To fulfill the program’s purpose, we will be asked to insert two inputs that will define the two polynomials we want to implement our operations on. After inserting them, we will need to choose between the six operations (addition, subtraction, multiplication, division, derivation, integration) that our program knows how to resolve. Each operation will be implemented as a method in the class Polynomial, but I will expand this subject later in my documentation. To show better how the program is supposed to work, I will present some use-case diagrams next (they will be modeled as lists, showing the steps involved in the execution of each case);

Use Case: operation using two polynomials

Primary Actor: Person using the application

Main Success Scenario:

-The person opens the application

-The person introduces two polynomials

-The application checks if the polynomial form of the input is valid

-The person chooses the operation desired to pe performed on the two polynomials inserted

-The program displays the result of the polynomial operation performed

-The person can select another operation next, using the same inputs, or insert new inputs and go through the process again

Use Case: operation using two polynomials

Primary Actor: Person using the application

Alternative Sequences:

-The input inserted does not follow the pattern implied for a polynomial

* The application will display a message that says the input is not a correct one: “Bad Input”
* The person using the program should delete the present input and give a new one; this means we are back at step 2

1. Project design (design decisions, UML diagrams, data structures, classes design, relationships, interfaces, packages, algorithms, graphical user interface)

In the following chapter I will discuss how I have split the problem into an object oriented one and the data structures I used for implementing the project, alongside the UML diagrams specific to this application.

Because we need to work with polynomials, I chose to implement two different classes that would help in my quest: one named Monomial and one named Polynomial. I will detail the variables and the methods of each of these classes in the next chapter, but I will quickly give an overview as to why I have chosen these classes and how I intended to use them further. The Monomial class will represent a singular monom object, having a double coefficient and an integer degree (we were previously instructed we would only work with polynomials of one variable and integer coefficients, as it is required). An easy example of an object contained in this class is the following: 3.5x^2; here, the coefficient will be 3.5 and the degree will be 2. Using the objects that we can create using this class, we will next implement the class Polynomial which will take as variable a List (moreover an ArrayList) of objects of class Monomial, which will basically construct the whole polynomial.

For example, if we have the following List: 2x^5, 3.6x^2, -9x^1, we will actually have the following polynomial: 2x^5+3.6x^2-9x^1.

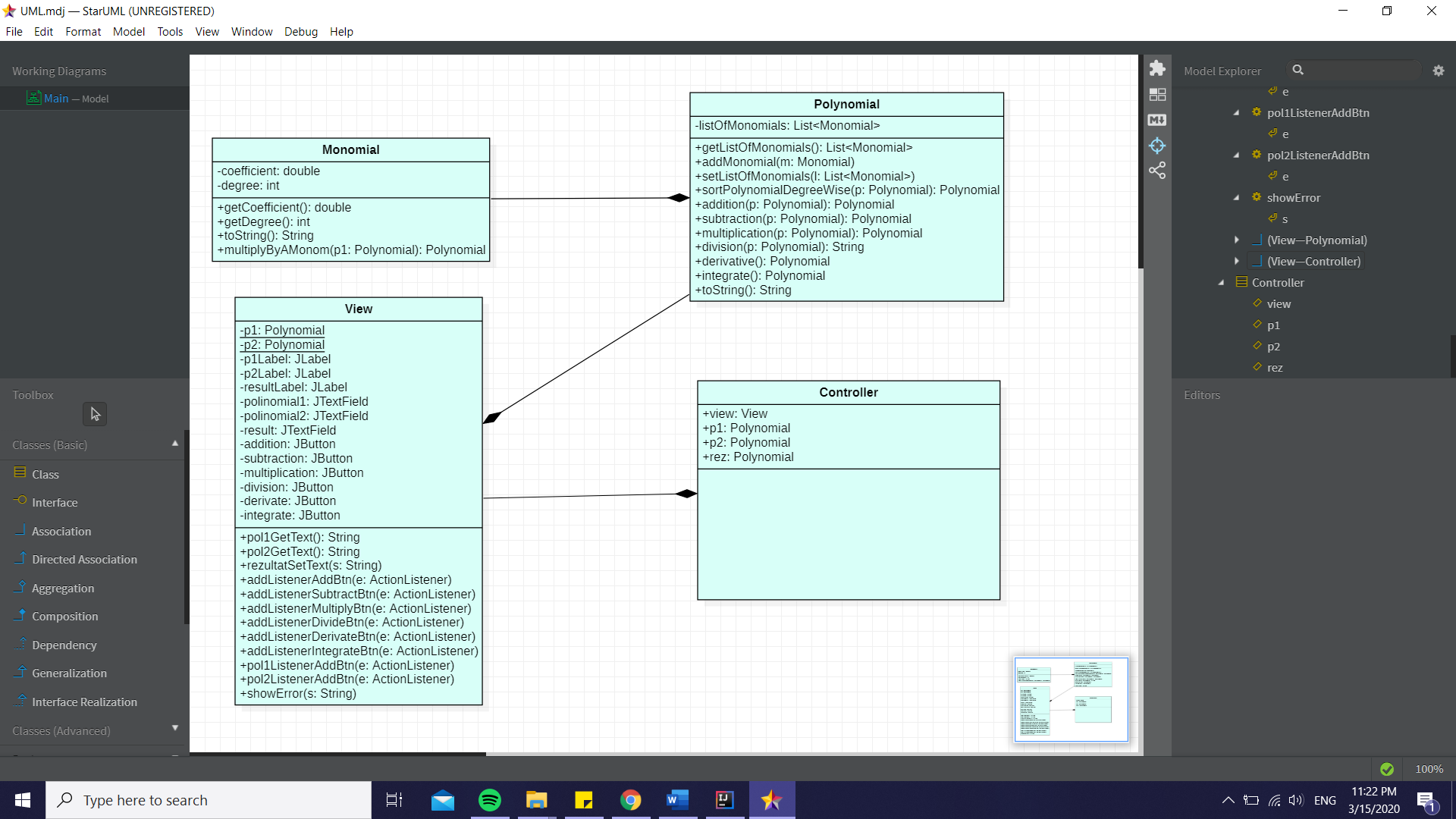
As for the package organization, I chose to implement 4 different packages, all for different purposes:

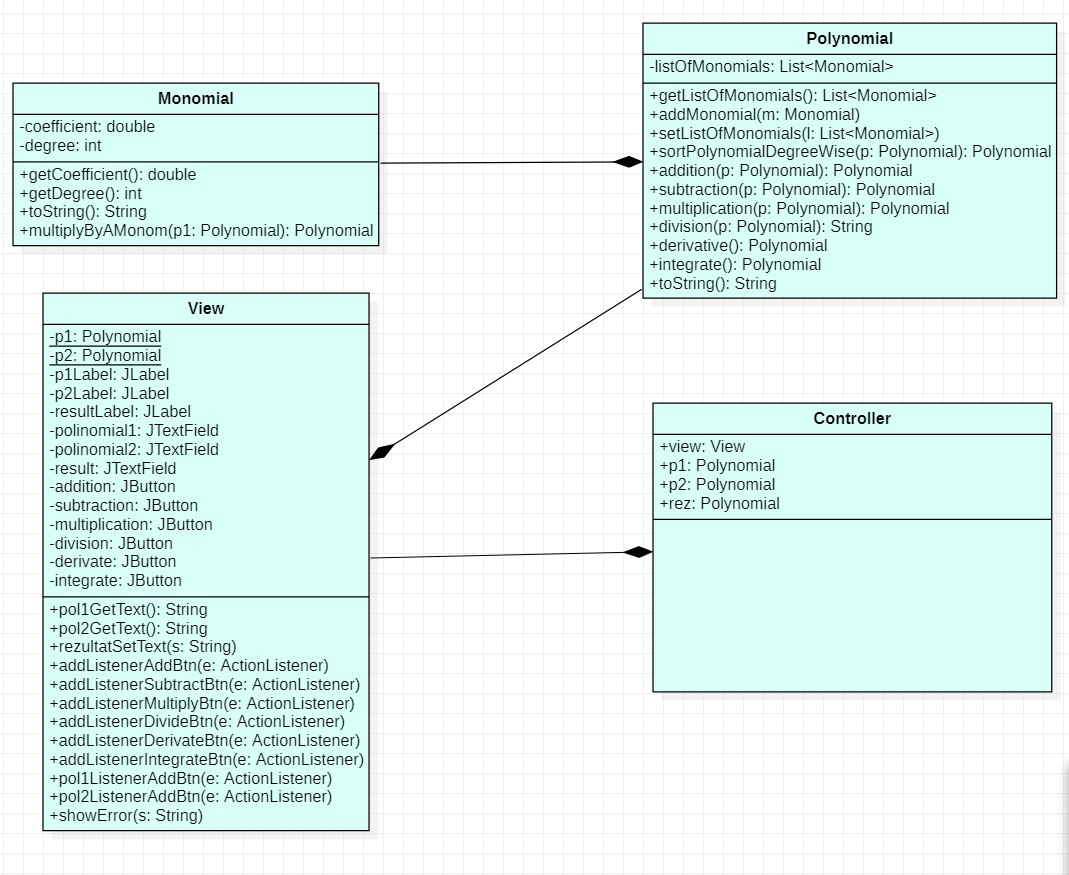
-Main: will contain the functions that will trigger the activation of the Graphical User Interface

-Model: will contain the two classes I presented earlier; this will be the implemented code that will do the actual operations on polynomials

-View: will contain the overview of the way the Graphical User Interface should look, the components that it will have and how will they be organized on the panel

-Controller: will contain the way the components described in the View package will be linked with the functions implemented in the Model package, giving real functionality to our user-friendly interface

Next, I will attach the UML diagrams specific to my project, along with the relationships between them (the following UML diagram is made using the StarUML app):



For making sure the inputs that are given by the user are in the correct form, I used Regex to verify the input. More exact, I used it when I overwrote Polynomial’s constructor. This constructor takes as parameter a String and construct the Polynomial object. Firstly, it creates a new ArrayList<Monomial>; then, it will take the following pattern: (-?\d\.\d+)[xX]\^(-?\d+) ; this basically means that we are looking for a format having a double number before x^/X^ and an integer after it. Because of this pattern, when we use the application, we will only have the right to give as inputs polynomials that have the coefficient as double. That’s why, even if we would want to write the simple polynomial 2x^2 we would actually need to write it as 2.0x^2. This part of the project I think could be improved, for a better use of the GUI, so that we could introduce both integer and double coefficients.

Next, we will take the pattern that we have and we will see if the input given matches it; if so, we will take the first group of the pattern, which is the coefficient and put it in a new double variable, doing the same thing with the degree group, only we are putting this one in a int variable, then we apply these variables to a new object of type Monomial. We then add this new monomial to the list assigned to our polynomial.

Lastly, in order to verify that our application works the correct way, I used JUnit to make a few tests on the implemented operations. For that, I added JUnit in the dependencies list of my maven project and I created a new class in the test part of the project. Then, for every single operation (except for division which isn’t working properly with the GUI, but I will touch on that in the next chapter) I wrote a test to show that my program works correctly. For example, for addition I gave the first polynomial p1 the value 2.0x^2 and the second polynomial p2 the value 4.0x^1+5.0x^0. Then, using the function Assert.assertTrue(p1.addition(p2).toString().equals("+2x^2+4x^1+5x^0")) we will test if the addition between the two inserted polynomials indeed is +2x^2+4x^1+5x^0. If that is the case, our test will give a positive result. We will do the same thing for all the operations implemented and the desired result is to have a positive test to all of them.

1. Implementation

In this chapter I will elaborate on each class I created, explaining the most important methods that each implement and why I chose them the way I did.

* Monom class

The objects that will be created from this class will help us further in implementing a full polynomial, as I already showed in the previous chapter of the documentation. The main methods that this class implements are the overwritten toString method and multiplyByAMonom.

Firstly, I overwrote the method toString because I needed a polynomial to be written as coefficient, x^, degree. That looks like the following: this.coefficient + "x^" + this.degree; The shown way, though, only works for integer coefficients (if the coefficient value is equal to (int) coefficient value), that’s why I decided to use Decimal Format to represent the case when the coefficient is of type double. For that, I declared a private static DecimalFormat df = new DecimalFormat("0.00"), which will show my double type variable only with their first two decimal points.

Following, I implemented the method multiplyByAMonom which has a parameter of type Polynomial and returns a Polynomial as well. This method will come in handy when we will describe the division method later. What the method does is it takes each monomial from the given polynomial and

* If the degree of the taken monomial is different from 0, then create a new monomial that has the coefficient=multiplication of coefficients and the degree=sum of degrees and add the monomial to the final solution
* If the degree of the taken polynomial is equal to 0 and the degree of our monomial from the class is different from 0, create a new monomial that has coefficient as the one before but the degree=degree+1; also add this new monomial to the solution
* If both the degrees are 0, then the new monomial will have the coefficient as the ones before it, but the degree equal to 0; add the monomial to the solution
* Polynomial class

Using the previously defined class, we will now construct the polynomials. The following methods are of most importance regarding the implementation of this particular class:

* sortPolynomialDegreeWise, which is used to arrange the polynomial in descending order of the degrees, which will be helpful when we will write de division method.

The method gets as parameter a Polynomial object and it returns the same polynomial, but “sorted”. For that we use the compare method which returns the difference between the degrees of two monomials given as parameters. This returned value is used by a Comparator object that is the parameter of the .sort method of the ArrayList that constructs the polynomial.

* addition, which will implement the addition of two polynomials.

The method gets as parameter a polynomial which will be added to the polynomial that implements the method and the result will be stored in a new polynomial that is later returned. For implementing the method, we go through the monomials that make up the list of the given polynomial and for the ones that have the same degree as any of the monomial making up the list of the polynomial implementing the function, we will add to the resulting polynomial a monomial having that exact degree and the sum of the coefficients. But what happens if one monomial from our parameter doesn’t find a corresponding degree in the one calling the method? For that I used a Boolean variable OK, which will change its value when a corresponding degree is found. If it is not found and the value of OK is still true, like in the beginning, then I will add the current monomial to the result. Also, to ensure the opposite also (when there is a monomial in the polynomial calling the function that doesn’t find a corresponding degree), I used another Boolean variable OK1. Going through both monomial lists, I check is the degrees are the same and if they are, I change OK1’s value. In case a degree is not found in the comparison, I add that monomial to the result too. This way, I can ensure that the addition is correctly written.

* subtraction, which will implement the difference of two polynomials.

The principle of this operation is the same as for the addition. We look for the coefficients which have the same degree and subtract them. Also, using a Boolean OK variable we check to see if all the monomials from the first polynomials have been involved in a subtraction, and if not, we add that monomial to the result. We need, at the same time, to add to the result all the monomials from the second polynomial that have not been involved in a subtraction. For that, similarly with the addition method, I used a Boolean OK1 variable and each time I found a monomial which hasn’t been subtracted, I add it to the result, but having its coefficient’s sign changed.

* multiplication, which will get as result the multiplication of two polynomials.

To implement this method, I used three new polynomials, one that will hold the final result, one for the elements that should be deleted from the result, and one for the elements that should be added to the result (I will detail this a few phrases later). Firstly, I went through both of the polynomials that are to be multiplied and I created the result from each monomial multiplied with each other monomial. This should be the equivalent of multiplying two parenthesis, term with term. But now, we will get a long sum containing, most probably more terms with the same degree that should be further added. For that, I go through the list of monomials describing the result twice (using two identical for loops) and, every time the two monomials are not identical and they have the same degree, I add them, creating a new monomial that is later added to the result, so I store it in the polynomial named polynomialsSumToBeAdded. Also, we will not want the term that we just added to still be in the final result, so I add them both in the polynomial I named polynomialsToBeDeleted. After going through all the polynomial result and adding everything that has the same degree, I will remove the monomials that are in polynomialsToBeDeleted from the result and also add the ones in polynomialsSumToBeAdded. For adding the correct ones, I go through only half of the list, because when we add them, there will be duplicates.

* division, which will give as result the division of two polynomials, as a String this time (I will explain later why as a String and not as a Polynomial like all the rest).

For division we use the previously implemented multiplication, and we will see exactly where in the process. For the division process to be implemented correctly we will need to sort the polynomial that calls the function degree-wise (actually we will clone that polynomial and arrange its clone). Also, I created two new polynomials, rest and result. At the same time, I stored in a new Monomial m1 the leading monomial of the polynomial that should be the divided by. I used an auxiliary int variable nr to keeps track of the degree of the monomial I should divide to (nr=0 initially). Then, while such a degree exists in the polynomial I mean to divide and while the difference of degrees between the monomial with degree nr and the leading monomial from the other polynomial is greater than 0, do the following: Add to the resulting polynomial a monomial with the coefficient=division on coefficients and degree=subtraction of degrees; multiply the same monomial with the polynomial that is the divisor (this is where the method multiplyByAMonom helps); subtract this product from the initial polynomial and set that value in the clone of the polynomial first given (this means we will continue to work with the new value, after subtraction); also, make sure you update the rest with the current value of the clone (when we could not make any operation of the ones mentioned above, this will be the final rest of the division).

Because we can have this rest, we should write the result as a string: when the rest’s list of monomials is not empty, the result should be written as: result.toString() + "+("+ rest.toString() + ")/(" + polynomial1.toString() + ")" (result+rest/divisor); otherwise, the method should return result.toString();

Even though the method works perfectly fine without the user interface (on the console’s output), I couldn’t exactly figure out why it wasn’t working when I linked it to the GUI functions. This too could be something this project could be improved on.

* derivative, which returns the derivate polynomials of the input.

This method is quite simple, as it adds to the result, for each monomial in the list of the given polynomial, a monomial having the coefficient=coefficient\*degree and the degree=degree-1.

* integrate, which is the opposite of derivate and it return the integration of a polynomial.

This too is a simple operation; it adds to the result a monomial (for each one in polynomial’s list) that has the coefficient=coefficient/(degree+1) and the degree=degree+1;

* View + Controller classes

I chose to put these two classes together because they both represent the implementation of my Graphical User Interface, so it made sense they would be explained together, since they are strongly linked with each other. Before I go on an explain each method separately, I wanted to say that my GUI is strongly inspired by the one showed in Ms. Pop presentation (the simple calculator) as well as by Mr. Giosan’s GUI examples from the OOP course (lectures 10 & 11), but I will also mention this in the bibliography part of my documentation.

As I previously mentioned, View class had the ‘appearance’ side and Controller class has the ‘linking’ side. But they both work together to give us the desired Graphical User Interface.

The View class has the following methods defined:

* pol1GetText & pol2GetText, which gets the text that is introduced by the user into the text field of our GUI and it is later user in the Controller part to attribute it to our input polynomials that we want to make operations on.
* rezultatSetText, which sets a new text into the result field of our GUI; it will receive the value of the resulting polynomial after the operation is performed.
* showError, which allows us to show an error message if the input is not typed in according to our predefined pattern.
* pol1ListenerAddBtn, pol2ListenerAddBtn, addListenerAddBtn, addListenerSubtractBtn, addListenerMultiplyBtn, addListenerDivideBtn, addListenerDerivateBtn, addListenerIntegrateBtn; all of these methods allow us to take actions when we click on different parts of our GUI(namely on the input fields, so we can write down the input, and on the operation buttons, so we can select which operation we would like to perform).

Also, I need firstly to define all of the variables I declared, doing that in the constructor part of the class. There, besides the obvious declarations of the text fields and buttons, I chose to implement EXIT\_ON\_CLOSE as default operation if you want to exit the GUI, as well as setting the content pane layout to null; this way I could arrange my text fields and buttons as I wanted, just specifying their place on an xOy axis along with their height and width. Lastly, I set the view panel to be visible for the user.

The Controller class, even if it doesn’t have specific methods, it does something else which is very important for a user interface: it implements subclasses that extend the ActionListener class, giving functionality to our previously defined GUI components. After defining everything in the constructor part (an example would be: view.addListenerAddBtn(new Add()) ), the class goes on to expand the meaning of each class. It should be specified that each of these classes have implemented the method

public void actionPerformed(ActionEvent e) using a try-catch method, in case the input is not well defined or the resulting polynomial is empty, in that case displaying the chosen error message, “Bad Input”.

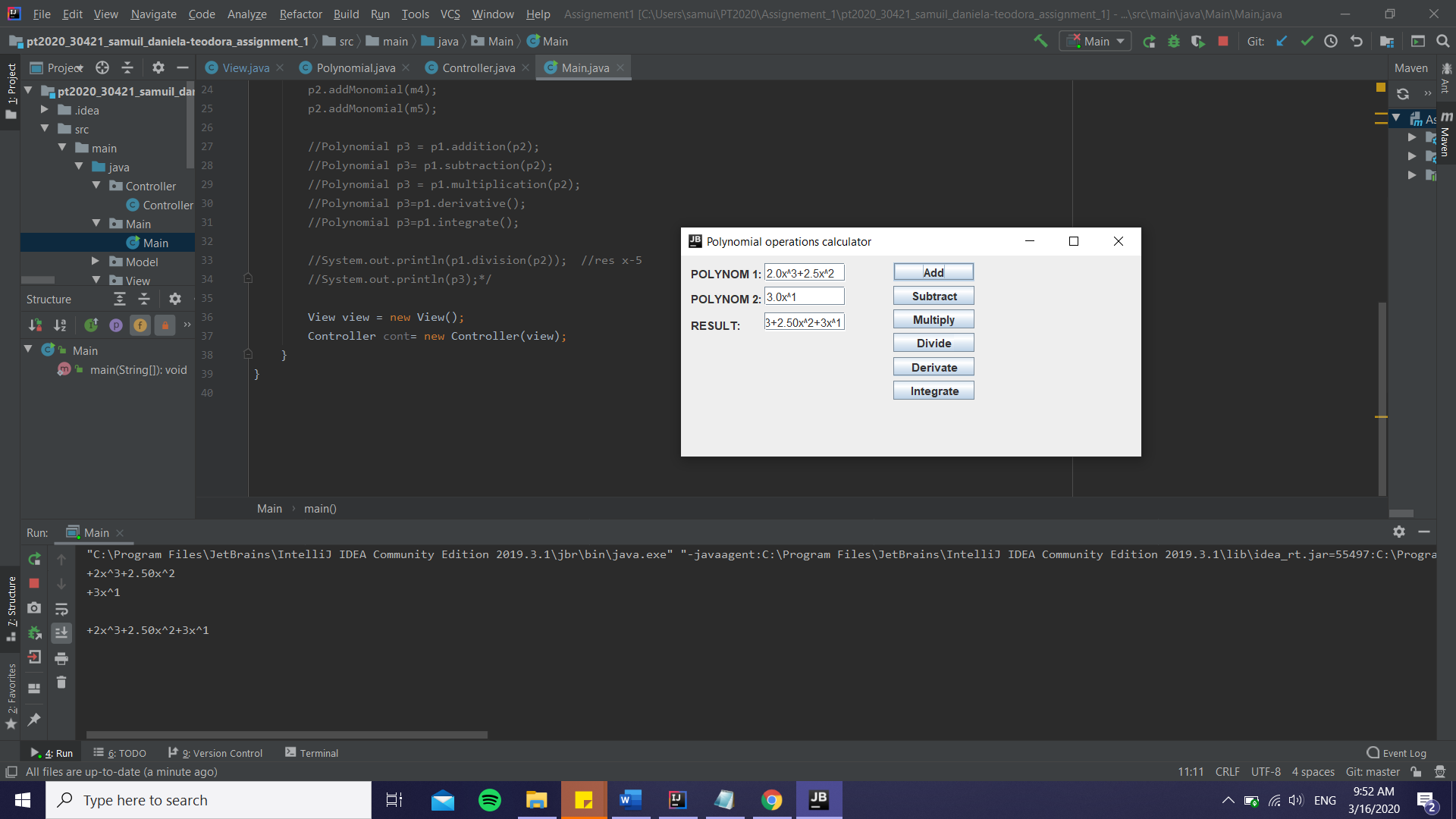
* class Polinom1Listener implements ActionListener & class Polinom2Listener implements ActionListener, which get the text from the input text fields and give the specific values to the input polynomials we will work with.
* class Add implements ActionListener, which gets the two inputs introduced by the user and makes the polynomial addition, displaying the result on the GUI result text field.
* class Subtract implements ActionListener, class Multiply implements ActionListener, class Divide implements ActionListener, class Derivate implements ActionListener, class Integrate implements ActionListener, which all work on the same principles as the Add class presented previously, only displaying different operation results, as their names suggest.

1. Results

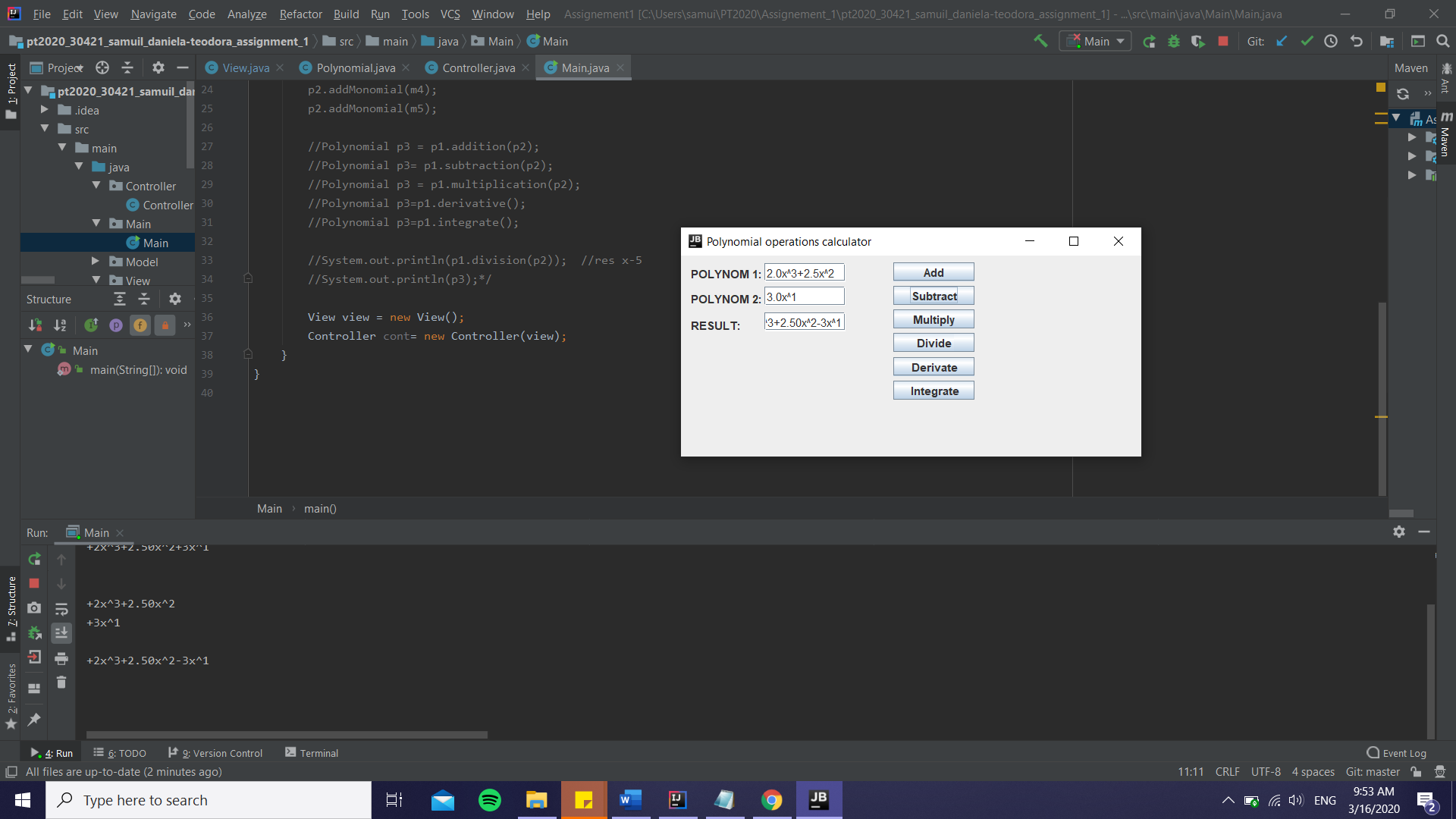
Even though I already touched on how the result will be displayed (I forgot to mention that I also implemented some System.out.printl() functions inside my Controller classes, to also display on the console both the inputs and the output after the operation is done, so a user can simply follow the calculations; I thought of this because I saw that, sometimes, for a ‘bigger’ input, the result in the text field could not properly show because it had too many terms (also something that could be improved, but I wanted the input text fields and output to be the same length for aesthetical reasons).

I will attach some screenshots of some of the cases using the GUI to show its functionality and the displayed results, both on the GUI surface as well as on the program’s console.

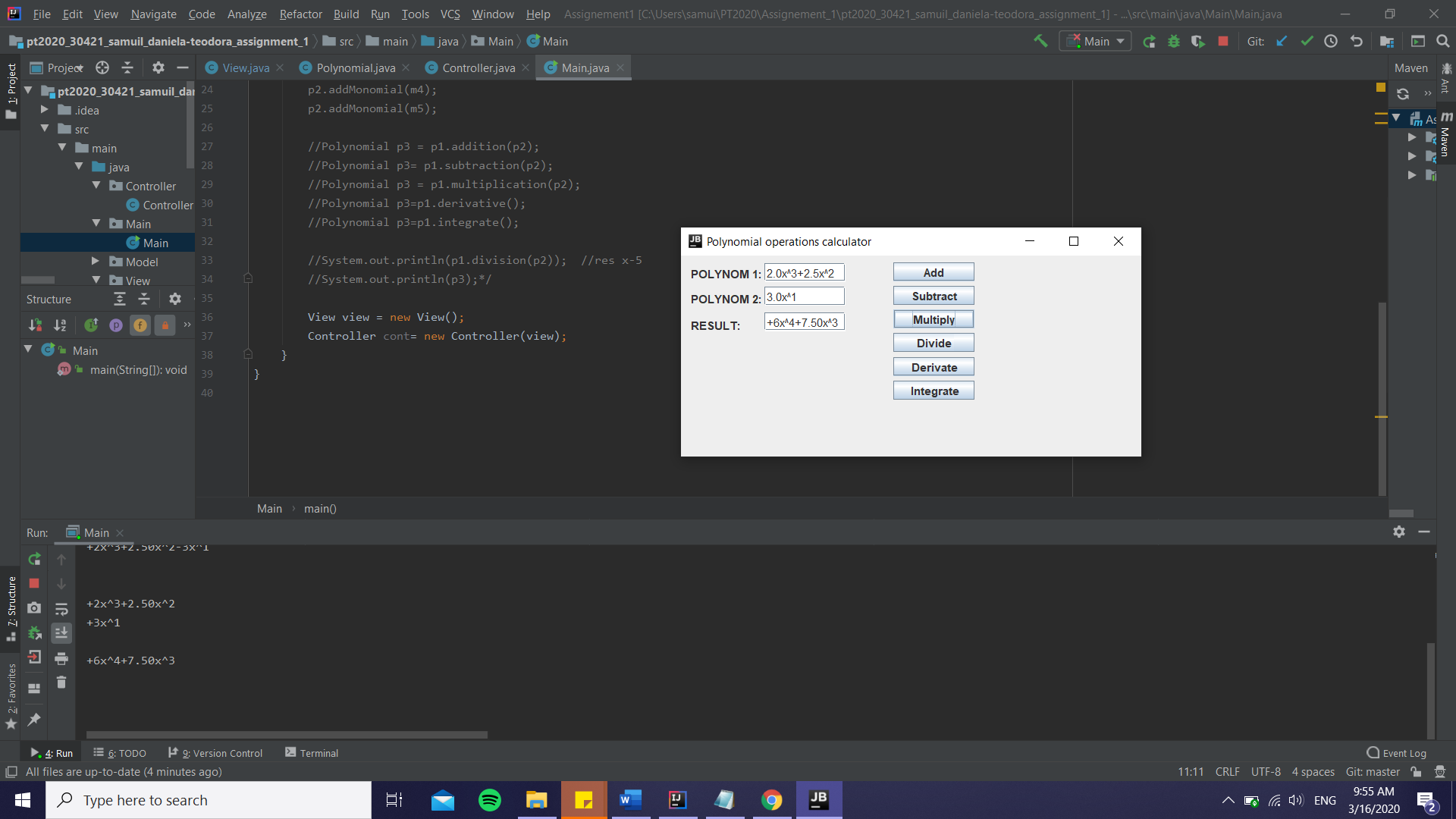
Addition:



Subtraction:

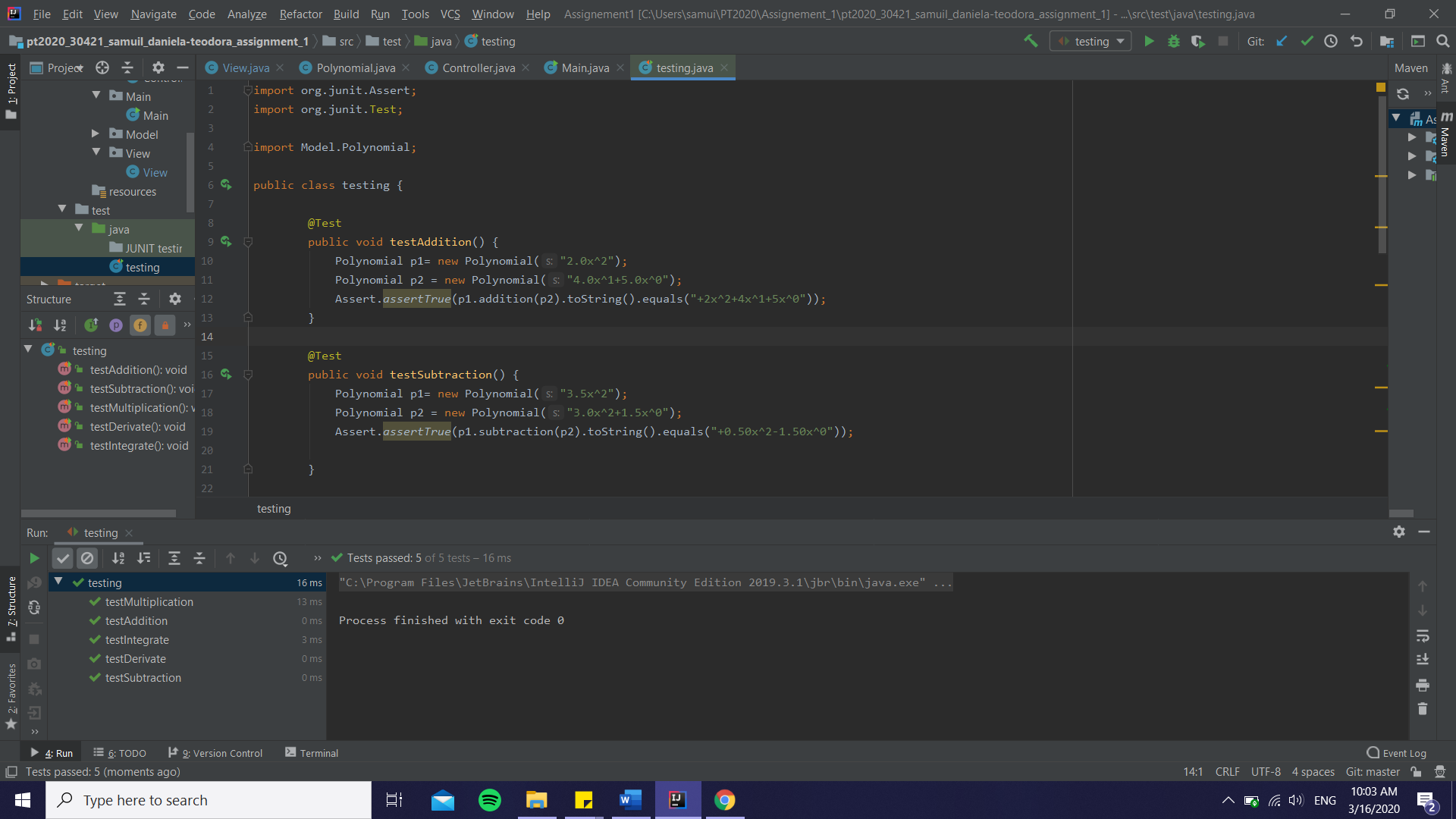


Multiplication:



The rest of the operations work the same way and they can be tested for within the project (except the division, which for some reason I couldn’t figure out, doesn’t display the good output when I use the GUI, even if it works perfectly without one (I tried this in the commented part of the main function); it displays the output with a rest, even if the division doesn’t have any).

Also, I will attach a screenshot of all the JUnit tests run (except for division), to furthermore prove the program is implemented correctly:



1. Conclusions

In conclusion, after finishing this project, I can definitely say I learned some new ways of verifying if an input follows a pattern(using Regex), test if my program works the correct way(using JUnit) and also solidify my Graphical User Interface knowledge, since I hadn’t had the occasion to work with is so often during the first semester. I think the requirements of this homework were very helpful to expand my knowledge of Object Oriented Programming and to furthermore learn new techniques which will come in handy in the future for sure (even the introduction to GitHub and bitbucket were something new and exciting, even if I found them a little confusing at first).

I certainly feel like my project could be later improved (I tried to touch on this as I went on presenting it earlier), especially the division part which seemed to have the most problems, at least for my kind of approach.

1. Bibliography

I will attach some links I felt were useful for my work within this project, as well as some mentions I thought I should make:

* I took great inspiration for the GUI model from Ms. Pop’s presentation <http://users.utcluj.ro/~crisb_pop/PTA1-SUPPORT-PRESENTATION.pdf> ,as well as from Mr. Giosan’s laboratory lectures from the first semester (especially laboratories 10 and a little bit of 11) <http://users.utcluj.ro/~igiosan/teaching_poo.html>
* For the sorting degree-wise algorithm, I used this link for further information: <https://www.geeksforgeeks.org/comparator-interface-java/>
* For more specifications regarding the use and implementation of the Decimal Format, I used the following:

<https://javarevisited.blogspot.com/2012/03/how-to-format-decimal-number-in-java.html>

* To help me remember correctly the polynomial long division, I used this online explication: <https://www.purplemath.com/modules/polydiv3.htm>
* For some more explications regarding the Regex pattern construction and how it should be used, I followed the link below: <https://www.vogella.com/tutorials/JavaRegularExpressions/article.html>
* To help me accommodate with the use of JUnit and everything it implies, I read the link below, as well as the presentation provided to us by our laboratory assistant, Ms. Pop: <https://mkyong.com/unittest/junit-4-tutorial-2-expected-exception-test/> ; <http://users.utcluj.ro/~crisb_pop/PTA1-SUPPORT-PRESENTATION.pdf>