

FACULTY OF AUTOMATION AND COMPUTER SCIENCE

FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 1: POLYNOMIAL CALCULATOR

Coordinator: Dr. Eng. Cristina Bianca Pop

Student: Bianca-Veronica Avram

2nd year, 2nd semester

2020

Contents

1. *Objectives*
2. *Problem analysis, modelling, scenarios, use case*
3. *Design*
4. *Implementation*
5. *Results*
6. *Conclusions*
7. *Bibliography*

*1.Objectives*

The requirement for this assignment includes designing an application for processing polynomials and effectuating different operations on them. The operations include:

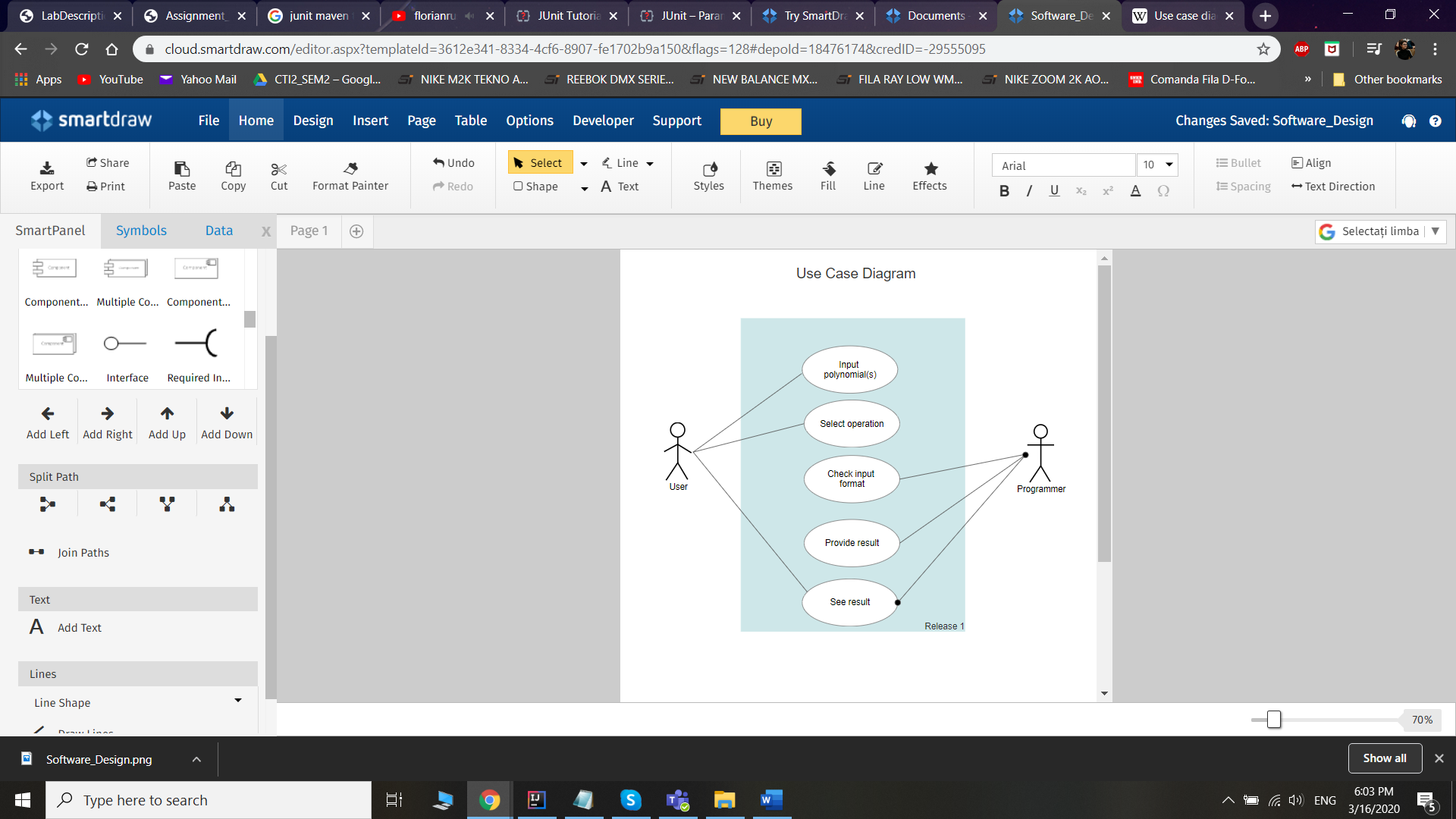
* Addition
* Subtraction
* Multiplication
* Division
* Derivation
* Integration

Alongside these operations, a Graphical User Interface (GUI) is required, such that the user is able to interact easier with the system. From this point of view, the objectives for this part include elaboration of the model-view-controller system in order to maintain a high level of clearness.

So, we can now state which are the secondary objectives of this assignment.

1. Drawing a conceptual model of the application
   * This part will be presented at point 3 of the contents and it consists a very important part in designing any application. The programmer should start from a less complicated representation of the solution to a problem, then go deeper with implementation only after the conceptual functionality and relationships are clear inside his/her mind
2. Identifying the classes and methods that should be implemented using oop principles
   * This may also be a part of the first objective, but more as a result from that. It represents the big step from the conceptual view to the actual preparation of implementation
3. Efficiently implementing all the classes
   * This part represents the coding part, where all the classes previously identified are implemented using java programming language, alongside all the methods that will be used in the purpose of achieving the correct functionality
   * Both 2 and 3 are included in point 4 of the contents
4. Designing a graphical user interface for the application
   * Described at point 4
   * This part consists of actually implementing the graphical user interface using java.swing
5. Establishing a well defined relationship between functionality and gui
   * Included in points 3 and 4
   * Here come the concepts of model, view and controller, which links the graphic interface to fulfill the requirements. The view is actually the user interface, the model contains the classes that give the functionality and the controller realizes the bond by implementing ActionListeners for all the buttons, associating the event of pressing a button in the view with the call of a method from the model
6. Processing the input data
   * Point 4
   * This part deals with validation of the input that the user provides. The polynomials are checked using Regex. Corresponding exceptions are thrown when the format is not followed
7. Testing for various inputs
   * Junit in parametrized tests
   * Point 5

*2.Problem analysis, modelling, scenarios, use case*



When the user runs the application a window (the main frame) opens and they can see 2 text fields for introducing polynomials, one (or 2) text fields which are read only where the result is provided and 2 sets of buttons. There is also a button to open the instructions which should be read before using in order to provide the correct input format.

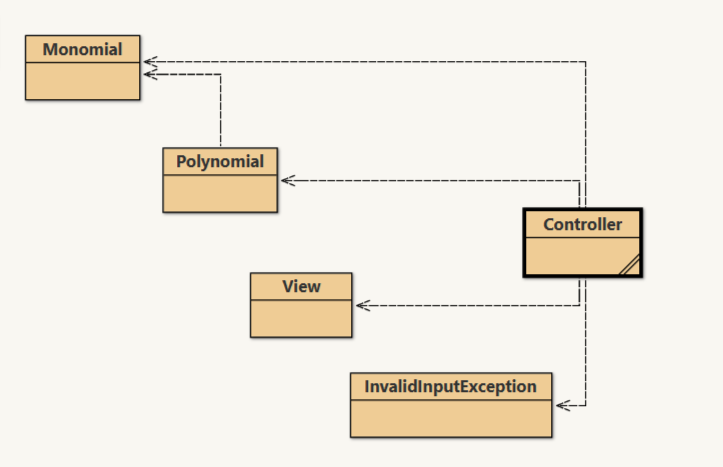
The user types the input data, then he/she clicks on one of the buttons. The buttons in the “Binary operations” panel require both fields to be completed. For the other set of buttons, “Unary operations”, only one text field completed is enough, corresponding to the button associated to each text field (derivate and integrate first are for the first text field and derivate and integrate second for the other one).

At this step, the user did his/her job. Now, the application checks the validity of the input data, which is done using the method getPolynomialFromTextField(String text). This method returns an object of class Polynomial if the input is in the correct format or it throws an InvalidInputException which pops up a message box telling the user that the input he/she provided does not follow the established format.

If the input is correct, then the ActionListeners that are attributed to the buttons make a method call in order to compute the result which is provided in the result text field. Note that for division there is a different format, as it requires two result fields. The second panel is made visible only when this operation is invoked. All the other times it is set not visible. Now the user can see the result and the cycle is completed. They can restart the process with nothing but introducing new polynomials as input and selecting another (or the same) operation.

Another remarkable aspect for division is when attempting to divide by 0 (the second polynomial is 0) a message box will tell the user that he/she is trying to divide by 0. This is actually an IllegalArgumentException which is thrown by the method that executes division.

*3.Design*



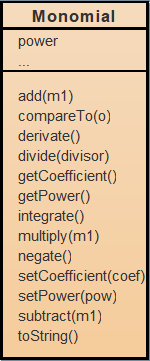
The application has the classes that can be noticed in the diagram above: Monomial, Polynomial, View, Controller and InvalidInputException, which is a user defined exception as a subclass of class Exception.

There are no defined interfaces, but Comparable and ActionListener were used. Comparable at Monomial, as the ArrayList of Monomials in class Polynomial are sorted, such that the highest power is the first one in the array. ActionListener is used in order to implement the method actionPerformed(ActionEvent e) for a button. Inner classes from the Controller class implement this interface.

For the ease of writing and avoiding multiple constructers or multiple classes, the coefficient of Monomial is of type Number, which makes it possible to convert the input int into double when necessary (namely when division or integration are performed).

In class Polynomial Collections are used, as the polynomial is actually an array of monomials, so it is defined as an ArrayList<Monomial>.

*4.Implementation*



The start point of this assignment is the class Monomial, which is a pair of coefficient and power and is the smallest part of a polynomial. The coefficient is of type Number for the reasons presented in the previous section. This class has 2 constructors, one parametrized and one without parameters, which initializes a monomial with power and coefficient 0.

As Monomial implements Comparable, in this class method compareTo() is overridden, such that the comparisons are made after the power field of Moonomial.

In this class all the operations are implemented, namely addition, subtraction, multiplication, division, derivation and integration, as the operations will be then extended in class Polynomial.

1. public Monomial add(Monomial m1)

This method takes as unique argument another monomial and returns a new monomial which is the sum of the monomial that calls the method and the monomial given as argument. This is done by computing the sum of the coefficients, as the powers are supposed to be equal. The method is only called for monomials with the same power (this is ensured by the implementation). The power of the result is the common power of the other two.

1. public Monomial subtract(Monomial m1)

This method uses exactly the same principle as the one given above. The only difference is that here the coefficients are subtracted, the one from the argument from the one calling the method. Again, the power of the result is the common power of the other two.

1. public Monomial multiply(Monomial m1)

This method takes as an argument a monomial and computes the product of the calling monomial and the argument. The coefficient is the product of the coefficients and the power is the sum of the two input powers. The result is a new monomial, having the previously discussed fields.

1. public Monomial derivate()

This method does not take any arguments and computes the derivative of a monomial. This is done by creating a new Monomial having the coefficient the product between the coefficient of the calling monomial and the power of the same monomial and the power decreased with one in contrast with the power of the calling monomial. An extra test is made here, in order to check if the calling monomial was actually a free term, which means that its derivative is 0. In this case, the result monomial is a new constructed monomial using the no parameter constructor (coefficient = 0 and power = 0).

1. public Monomial integrate()

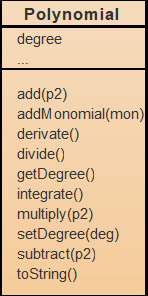
This method returns a new monomial representing the primitive of the calling monomial. The coefficient is the coefficient of the calling monomial divided by the power incremented with one. The power is the old power incremented with one. Here, the division is done by taking the doubleValue of the class Number. When converting to string, only 2 decimals are taken.

1. public Monomial divide(Monomial divisor)

This method is similar to multiply(), but here the coefficients are divided, which means that again doubleValue is used, and the power is the difference between the powers of the two input monomials.

1. public String toString()

This is the last important method that deserves to be presented here. It is @Overriden and it provides the printable version of a monomial in a format that the user understands, namely “[coefficient]x^[power]”. Further checking for the most elegant representation of the monomial is done by the toString() method implemented in Polynomial class.



This class gives the main object of the whole application, as the purpose is to work with polynomials. All the required operations are implemented in this class as methods, using, of course, the lower level operations on monomials. A Polynomial object has two fields: poly, which is an array of Monomial objects, and degree, which is the highest power of the monomials that make up the list. There is also an additional field given as another class, namely QRPair. This class is used for the division method, which needs to return 2 parameters for the whole result, so, the fields of QRPair are two Polynomials, quotient and remainder. The pincipal methods implemented will be presented in the following paragraphs.

1. public void addMonomial(Monomial mon)

This is the method that appends to the monomial list a new monomial, given as argument. This is done by using add() method of the class Collections.

1. public Polynomial add(Polynomial p2)

This is the method invoked for adding two polynomials, one calling the method and one given as argument. The result is a new polynomial that is formed by first copying the first polynomial, then going through the second one, searching matching exponents of monomials. If a match is found, the coefficient of the corresponding power monomial in the list is updated, using the add() method from Monomial. If 0 is obtained, than that monomial is removed from the list. If, after the sequential search of a matching power, that exponent is not found, then it is appended at the end of the list. Collections.sort(poly,Collections.reverseOrder()) is used to arrange the monomials in the list in decreasing order of powers. The degree of the resulted polynomial is set if there are still monomials in the list. If the list is empty (the added polynomials were opposite), then a polynomial having a unique monomial in the list (the one built with the no parameter constructor) is returned.

1. public Polynomial subtract(Polynomial p2)

This method works exactly like add() method, but the operation on monomials is subtract(). The rest is just like above.

1. public Polynomial derivate()

This method is based on the derivate() method from Monomial class and it works like this: the list of monomial is iterated sequentially using a foreach loop and each monomial is derivated and added into the result polynomial. The degree of the polynomial decreases by one. If the degree gets to -1, that means that the initial polynomial has only one single term, a constant, so the result must be 0. The returned polynomial in this case is one containing only the null monomial.

1. public Polynomial integrate()

This method id based on the integrate() method from class Monomial. The result polynomial is one containing all the monomials from the source polynomial integrated. The degree is updated to one larger with one.

1. public Polynomial multiply(Polynomial p2)

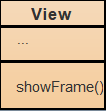
Honestly, I had troubles with this method, because it does not work properly, because the sign id inverted for all the resulted terms and I could not figure out why this happens, because I only have addition and monomial multiplication in this method, which are not supposed to have an effect on the signs. To fix this, I just built another result polynomial, where all the composite monomials take the opposite sign. **Update**: I found the problem and I fixed it. I do not need any additional variable. And, to describe the functioning mode of this method, it first checks if one of the polynomials is 0, so that the result will be directly 0. Otherwise, the algorithm takes each monomial from the first polynomial and creates a new polynomial having the results from multiplying the recently mentioned monomial with the monomials from the second polynomial. This polynomial is then added to a result polynomial, and the process is retaken for all the monomials in the first polynomial.

1. public QRPair divide(Polynomial p2)

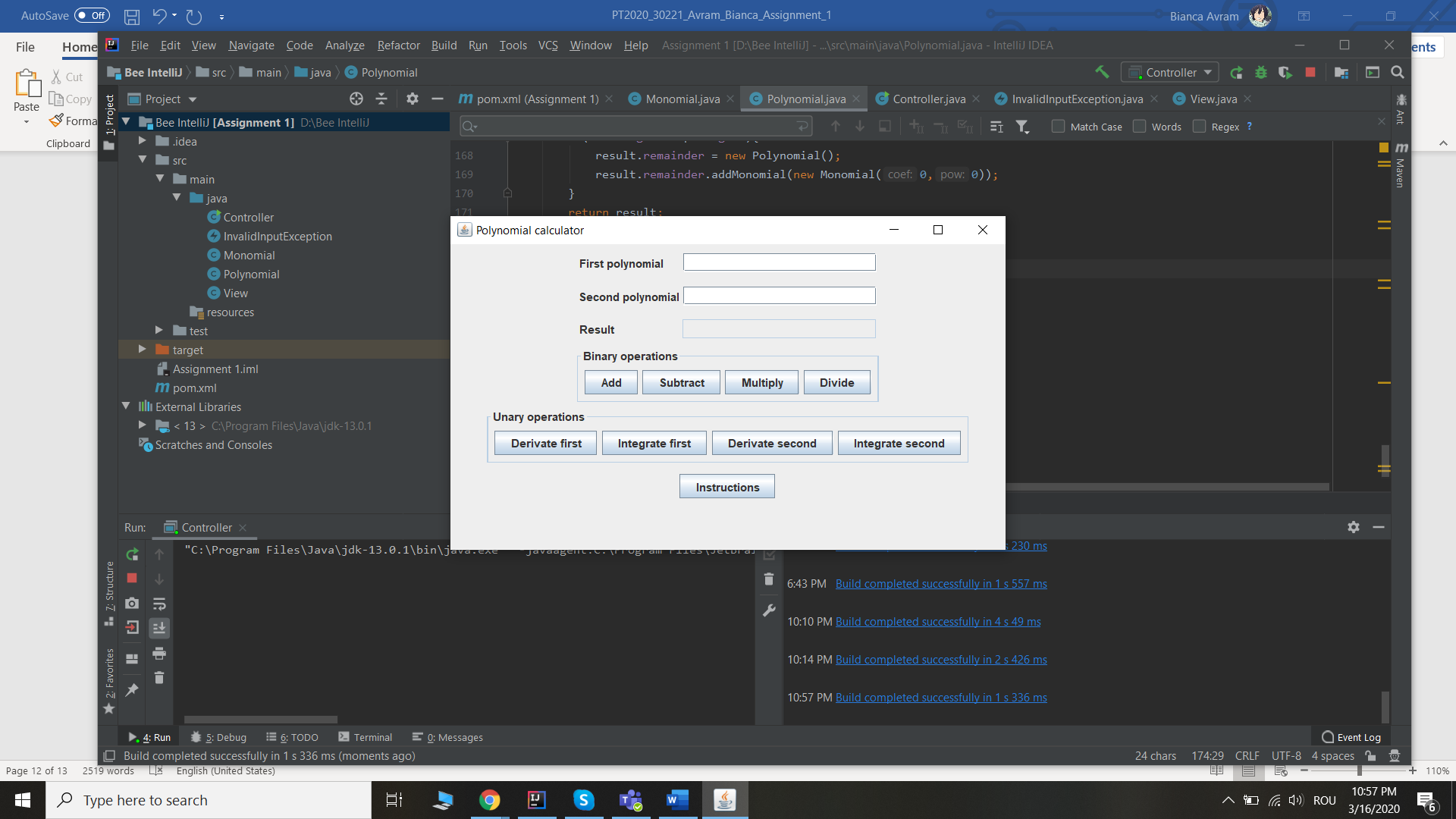
The divide method implements polynomial division between the polynomial that calls the method and the one passed as argument. The return type is QRPair, a class defined inside Polynomial class which contains two Polynomial fields, one for quotient and one for remainder. If the user attempts to divide by 0 an exception is thrown and he/she will be notified that the operation is illegal. The algorithm starts by copying the calling polynomial in the remainder field of result. While the degree of the remainder is greater or equal to the degree of the degree of the divider and the remainder is different from 0 the result from dividing the first monomial of the remainder by the first monomial of the divider is added to the quotient. Then this monomial is multiplied by all the monomials in the divider, then the result is subtracted from remainder. This process repeats until the while condition is not fulfilled anymore. If the degree of the first polynomial is smaller than the degree of the second one, then the result will have as quotient 0 and as remainder the first polynomial.

1. public String toString()

The last significant method here is the @Overriden toString() method which gives the string representation of the polynomial in a format for the user to understand it.



The class View implements the graphical user interface of the application. A frame is built, havnig 2 text fileds for input, where the user types the polynomials he/she wants to work with. There are other 2 text fields which are read only, where the result will appear after processing. One of them is set on visible false, because is used only for the division operaion that returns two polynomials as result.

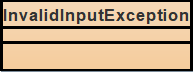


Then, on the frame there are 2 sets of buttons, one for the binary operations, that involve 2 input polynomials, and one for unary operations, that involve only one polynomial. The binary operations are addition, subtraction, multiplication and division, each of them having a corresponding button which initiates the performing of that operation, giving then the result in the result text field. The unary operations are derivation and integration, which I decided to implement twice, for both available text fields. So, derivate first and integrate first return the result for the polynomial in the first text field and derivate second and integrate second for the polynomial in the second text field.

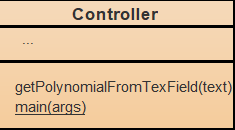
There is one extra button on the bottom that opens a message box containing the instructions for using the application properly.

This class only has a constructor and a method that instantiates a View type object and makes the frame visible and sets its attributes (dimensions, close operation, location).

For the whole graphic interface java.swing was used. Components are grouped in panels and then placed on the main frame.



The class InvalidInputException extends Exception and it is thrown when the polynomials the user wants to work with are not in the correct format specified in the instructions.



The class Controller is the piece of the program that links everything presented previously, including the user interface by implementing inner classes for ActionListeners that provide functionality to the application. Each button has its listener where the corresponding operation is executed and the result is provided.

This class contains the main method that runs the program. Here the ActionListeners are written and added to all the buttons to provide functionality.

The constructor instantiates a View object.

There is only one other method that is used to validate the input from the user. When a button is pressed, first the method getPolynomialFromTextField(String text) is called. This method first checks if the input is valid. An InvalidInputException is thrown if the input is in the wrong format and the user is asked to introduce other data. Otherwise, the corresponding operation is executed, as the methods from Polynomial are called. The result is obtained and then it is placed in the result text field(s).

In what concerns the validity of the input, Regex is used. Then, the valid polynomial string needs to be split into monomials and a polynomial is built, then used in further processes.

*5.Results*

I tried to run a test using JUnit Parameterized test, but it failed for both cases. Testing the application as a usual user succeeded, but JUnit seems a little bit too complicated for now for this problem. I built two test classes, PolynomialTest and MonomialTest and set them up for testing, but the results were not the wanted ones.

*6.Conclusions*

The main conclusion I can drop from this assignment is that I cannot use Junit, but this is completely my fault, as maybe I didn’t try hard enough. So, as further development I aim to implement unit testing using JUnit.

In what concerns the application, this is the first java project this semester, so I can admit that my skills were awakened and developed. Also, as mathematical knowledge was a little bit required in solving this task, those abilities from 12th grade came in to help.

The application is functional and can actually be used as a polynomial calculator, having all the required features. The graphic interface is easy to understand and the fact that there is an Instructions button is a helpful hand for the user.

As further development, introducing polynomials with floating point coefficient might be a worthful idea an also calculating the value of the polynomial in a certain point. Maybe Viete’s relations may be introduced to work with the coefficients or an algorithm for finding the roots of a polynomial.

The most difficult part of this assignment was the division, which is a more complex operation, having a separate algorithm.

*7.Bibliography*

1. <http://users.utcluj.ro/~crisb_pop/PTA1-SUPPORT-PRESENTATION.pdf>
2. <https://en.wikipedia.org/wiki/Polynomial_long_division>
3. <https://stackoverflow.com/questions/36490757/regex-for-polynomial-expression>
4. <https://docs.oracle.com/javase/7/docs/api/javax/swing/JOptionPane.html>
5. <https://docs.oracle.com/javase/tutorial/uiswing/components/frame.html>
6. <https://docs.oracle.com/javase/tutorial/uiswing/components/panel.html>