Technical University of Cluj-Napoca

Polynomial Application

Stefania Sucitu

Group 30421

March 2016

Abstract

The developed application serves many very useful purposes, which come in handy for any user who wishes to perform operations on polynomials. Based only on Java, the application offers a very user-friendly interface and a reliable structure of operations, which ensures the correctness of the performed calculations. The user has the possibility of typing in the desired polynomials, selecting an operation form the six of the provided and a result, which is correct according to tests will be displayed.

Introduction

The application in discussion comes in handy when handling polynomials and basic operations involving them. Mathematics and Computer Science, as well as many more fields of science and research, are very much tied to the polynomials and their functions. The task of working with polynomials may look simple at first, however, when looking at higher and higher degrees, doing operations by hand may cost one very much time, which is not always available and the calculations may become too difficult for one to solve by hand. The application is addressed to students who wish to perform basic operations on polynomials. It is especially aimed to mathematics and informatics students, as well as any other domain connected to the previously mentioned domains, such as physics, sciences and many more engineering domains.

Aim of the Application

The main objective of this mini-project is to develop an application which offers a simple to understand user interface which holds different operations on two polynomials, given by the user, of a certain degree. The user is asked to input two polynomials and then select which operation he wants the application to perform, all this followed by the display of the result, in a pleasant, appealing view. The main functions aimed to be performed are: addition, subtraction, multiplication, division, differentiation and integration.

One of the main features of this application is the window in which all these user-application interactions take place. It is desirable to have two different input areas for the polynomials, which will continue to be displayed along with the result of the chosen operation. What would be considered more important is the fact that the user should be able to use the interface given, without any help, in other words, the look of the window to be self-explanatory.

Problem Analysis

Modeling

The main aspect to be considered into the development of this particular problem would be the representation of the polynomials. For this, I have chosen to represent the polynomial as a series of monomials. To be more specific, the polynomials would be made of pairs of coefficients and degrees. Such pairs would give the final form of the polynomial.

In the program I have chosen to create two separate classes: *CoeffDegree*, which will stand for the creation of the previously discussed pairs and a class *Polynom* which will create the polynomials by arranging the pairs into an array list. Also this last mentioned class will implement a method which will convert the user input polynomial into a form which the program can work with.

The CoeffDegree class represents the backbone of the entire project. It holds two of the main attributes for the application, namely Coefficient( *coeff*) and Degree(*degree*). This class also implements getters and setters which will come in handy in the later development of the project

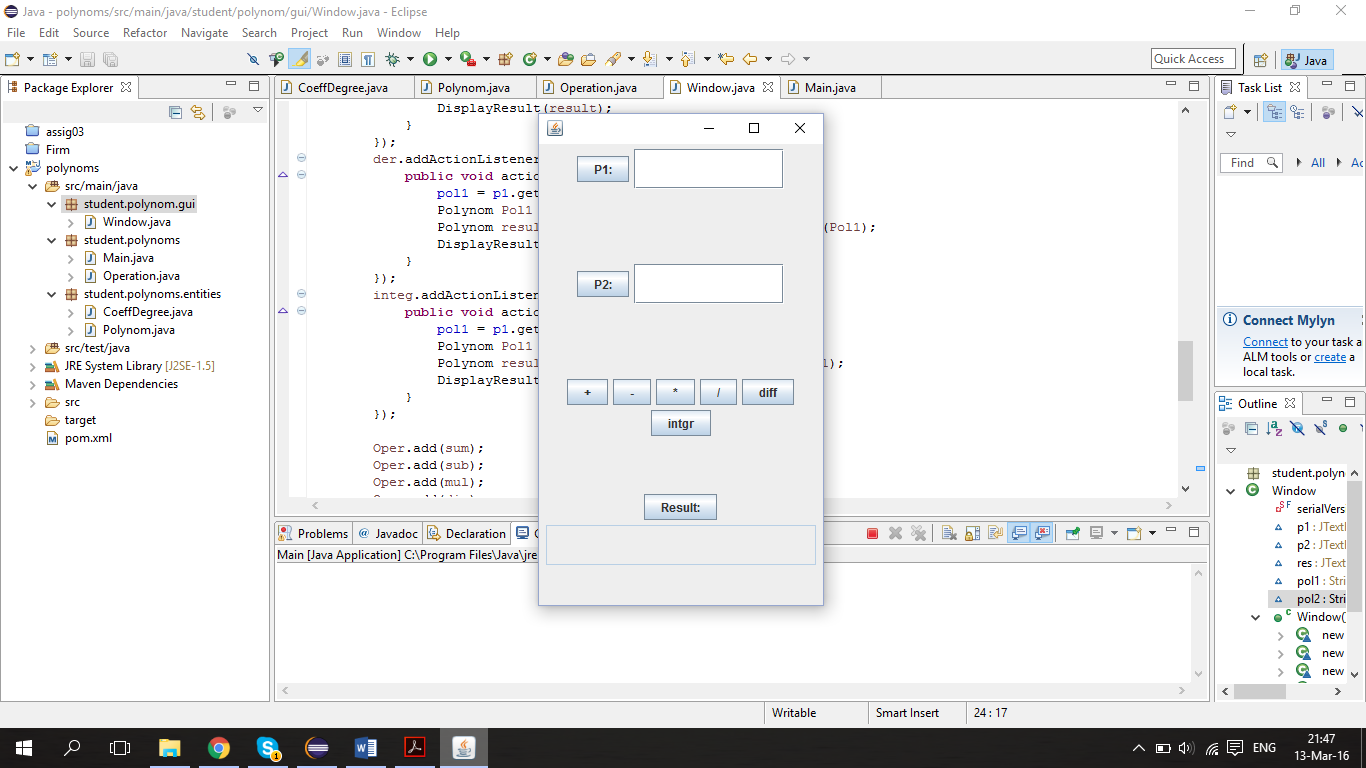
The *Polynom* class is a useful tool for the given problem. The main goal of this class is to create the further used polynomial, as an array list of coefficient and degree pairs. This class also implements a very important method, which performs the conversion of the user input to the format used by the program. Details of the implementation of this method will be given later.

Going further with our modeling process, we reach the point where we should determine what actions our previously created entities should be able to perform. For this step we should consider use cases. Let us take as an example the addition operation. First we should take into account the initial requirements, meaning having two separate polynomials, a trigger for the operation, in our case we will use a button with an action listener implemented, and a result filed. To fulfill these basic, initial requirements we need to model a window accordingly, using the swing library.

Next we need to check some steps for our operation to complete. The user should type in the specially created text box the polynomials and hit the trigger. The operations will be then performed and the result should be displayed instantly on the screen, in the dedicated for such area. In order to constantly check the correct flow of the execution, the console of the java runner program should be checked. If everything goes as it should be, meaning no errors, the nothing will be displayed.

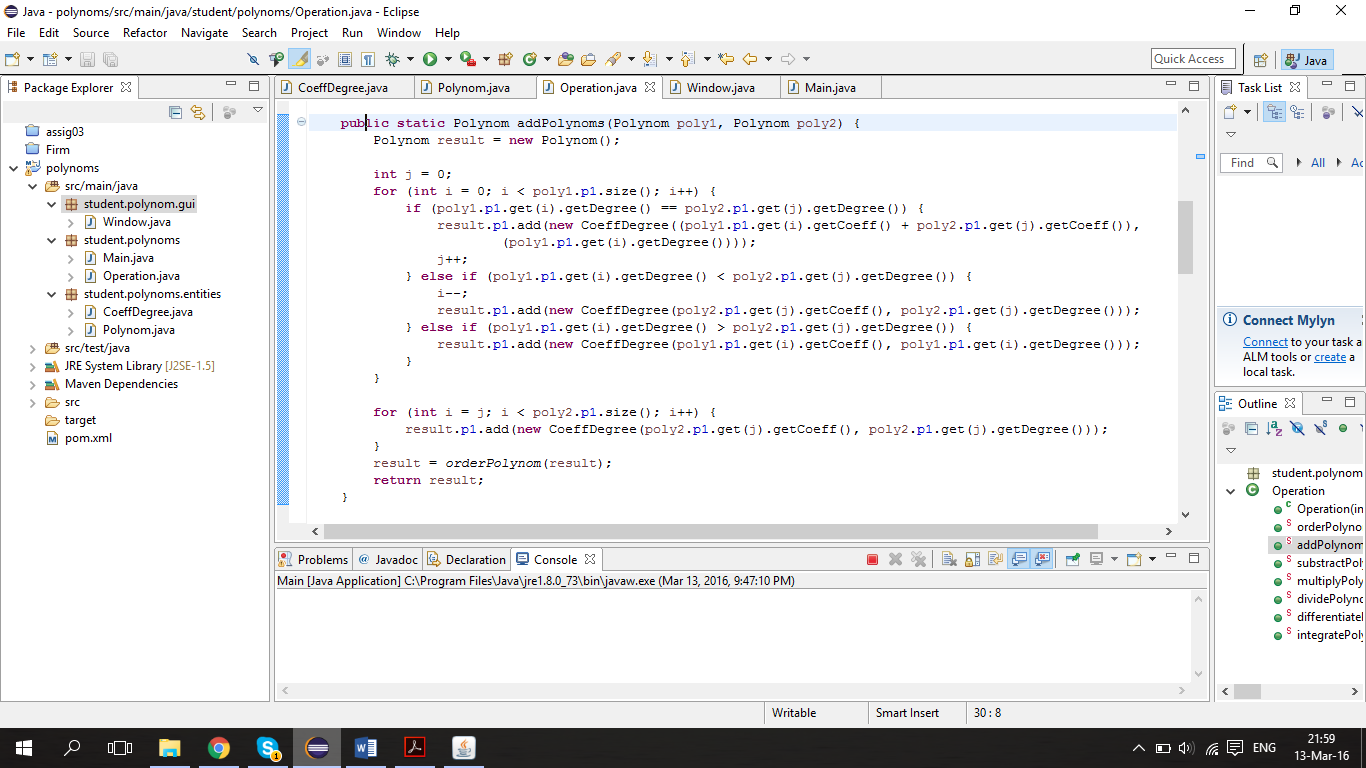
Taking into account the previously mentioned aspects, we should have an almost complete image of how our window should look like. Accordingly, a Window class is created, which implements and displays, with the help of swing elements the desired result.

The final aspect of the discussed window is shown below. As we can observe, there are separate input text boxes, foe each one of our polynomial, one text box which will display the result, according to the picked operation. The selection of operation is done by pressing one of the six available buttons which implement, in order, the following operations: addition, subtraction, multiplication, division, differentiation and integration.



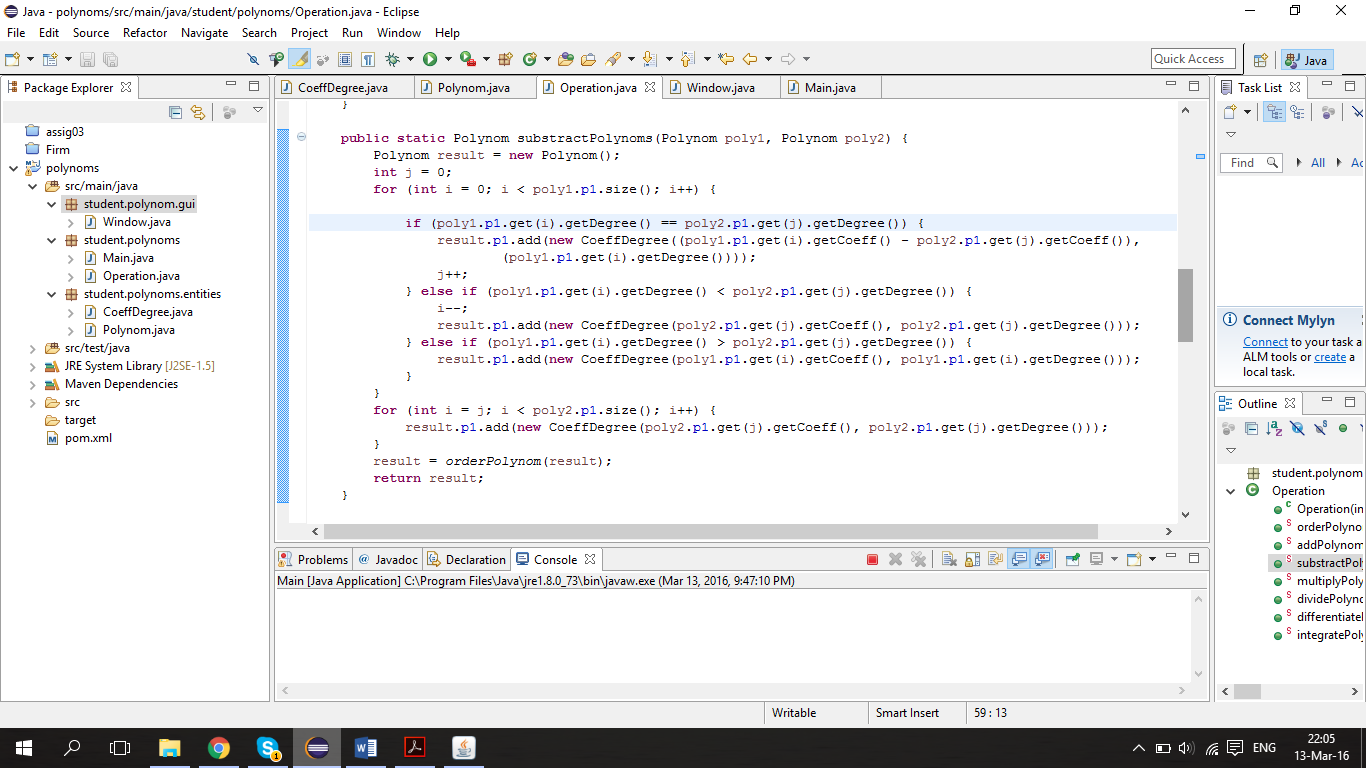
Besides the CoeffDegree class which was mentioned before as the backbone of the entire application, the main feature and most important class of this application is the Operation Class. The class in discussion gives sense to the project as it implements the operations mentioned above. In the next part I am going to present the implemented operations, offering a view of the code implementation, as well as a few explanations and, where required, the mathematical background of the operation.

1. Addition

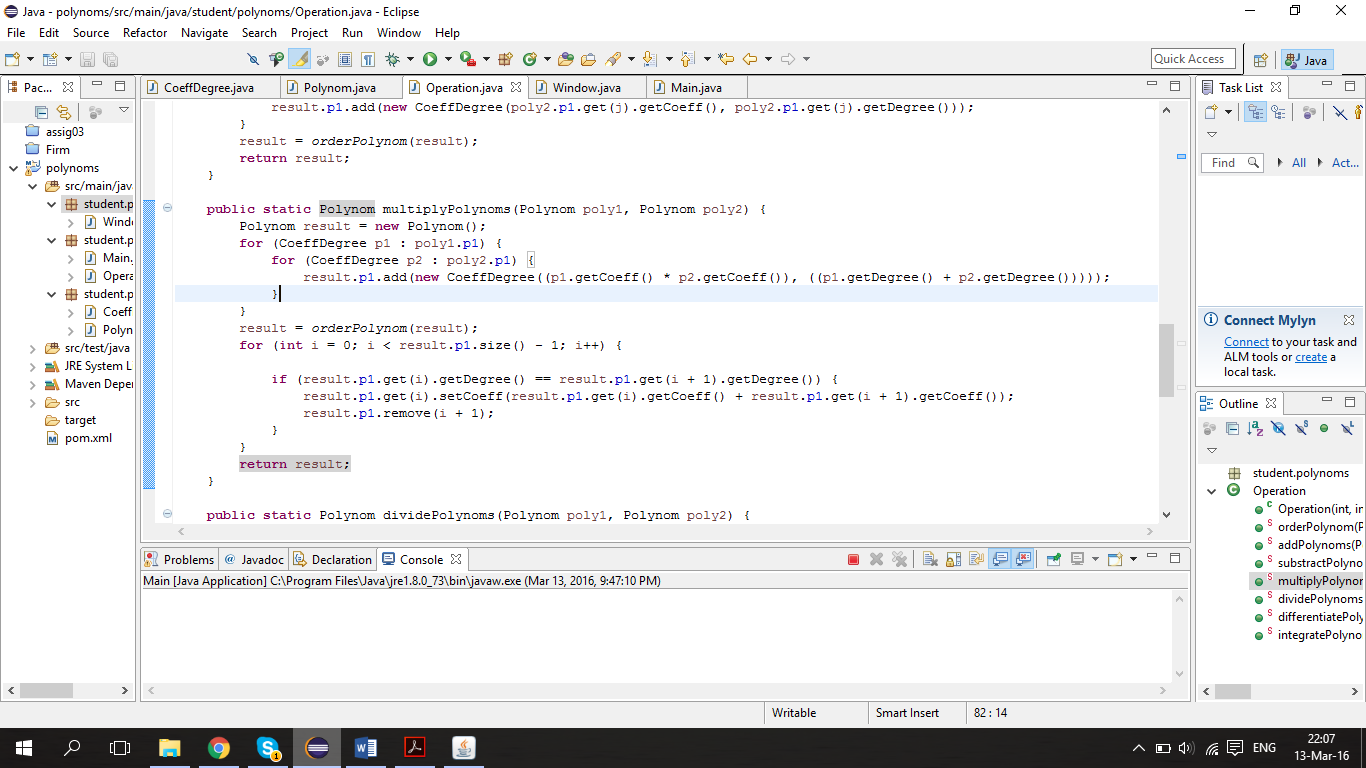


The basic algorithm behind the addition operation is to iterate through one polynomial and check it the elements, one by one matches the degree of the other polynomial. Based on the results of the comparison an element is added to the additional result polynomial as it follows: the added coefficient if the degrees match, only the first element if there is no such degree in the second element and only the second if the case is the same. We use an additional method to order the elements in the result polynomial so a more appealing result is obtained.

2. Subtraction

For the subtraction algorithm we use a similar approach to the addition, making only the necessary adjustments.

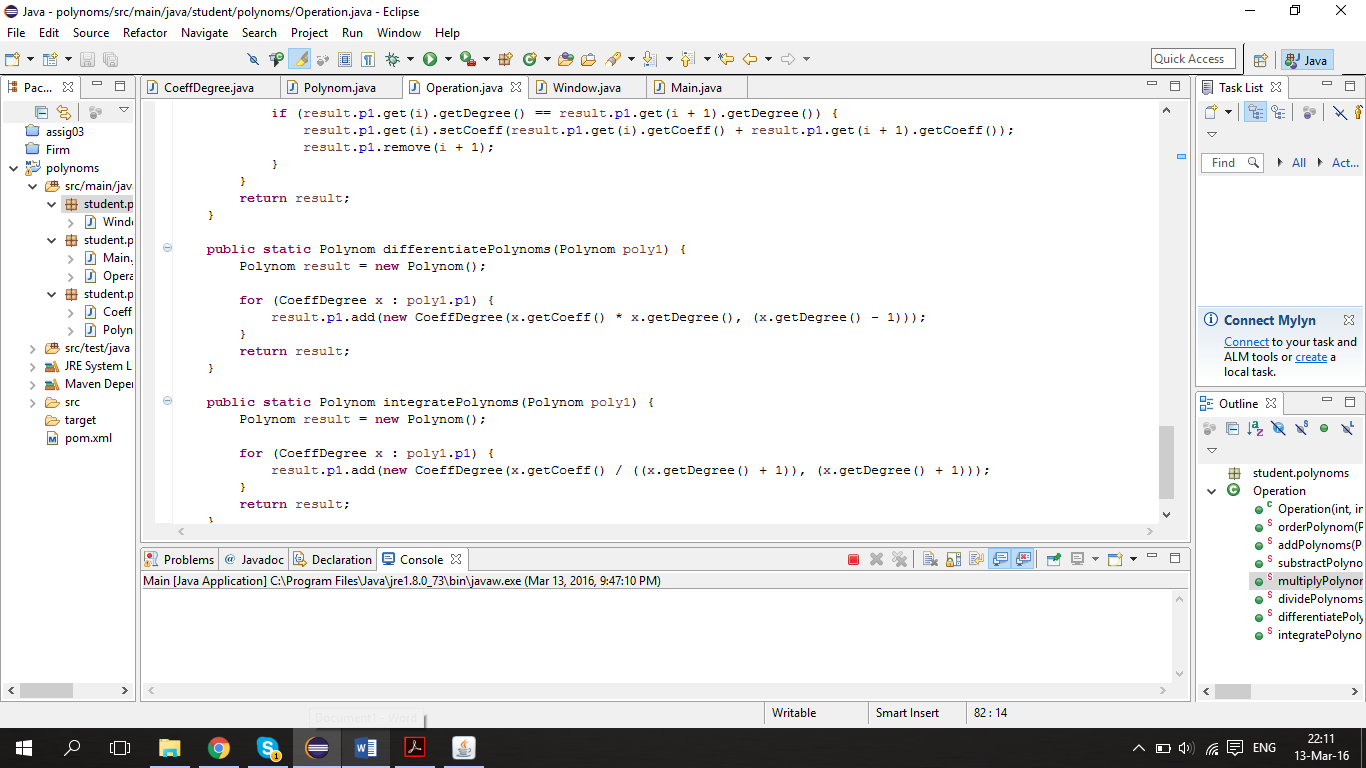
3. Multiplication



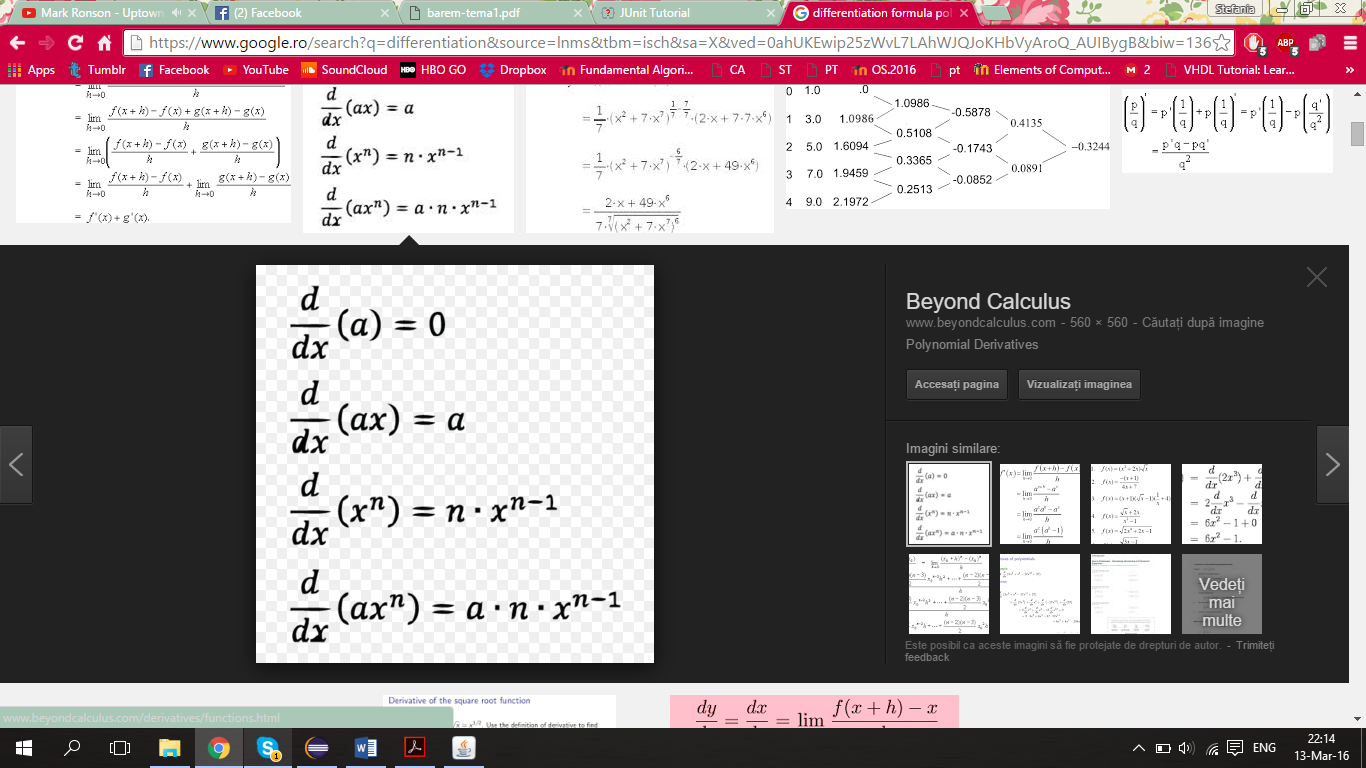
When implementing the multiplication operation we considered the basic multiplication of monomials, multiply the coefficients and add the degrees. This operation is done for each of the elements in the two polynomials and the resulted polynomial then goes through an addition operation for the elements with the same degrees.

4. Division

5. Differentiation

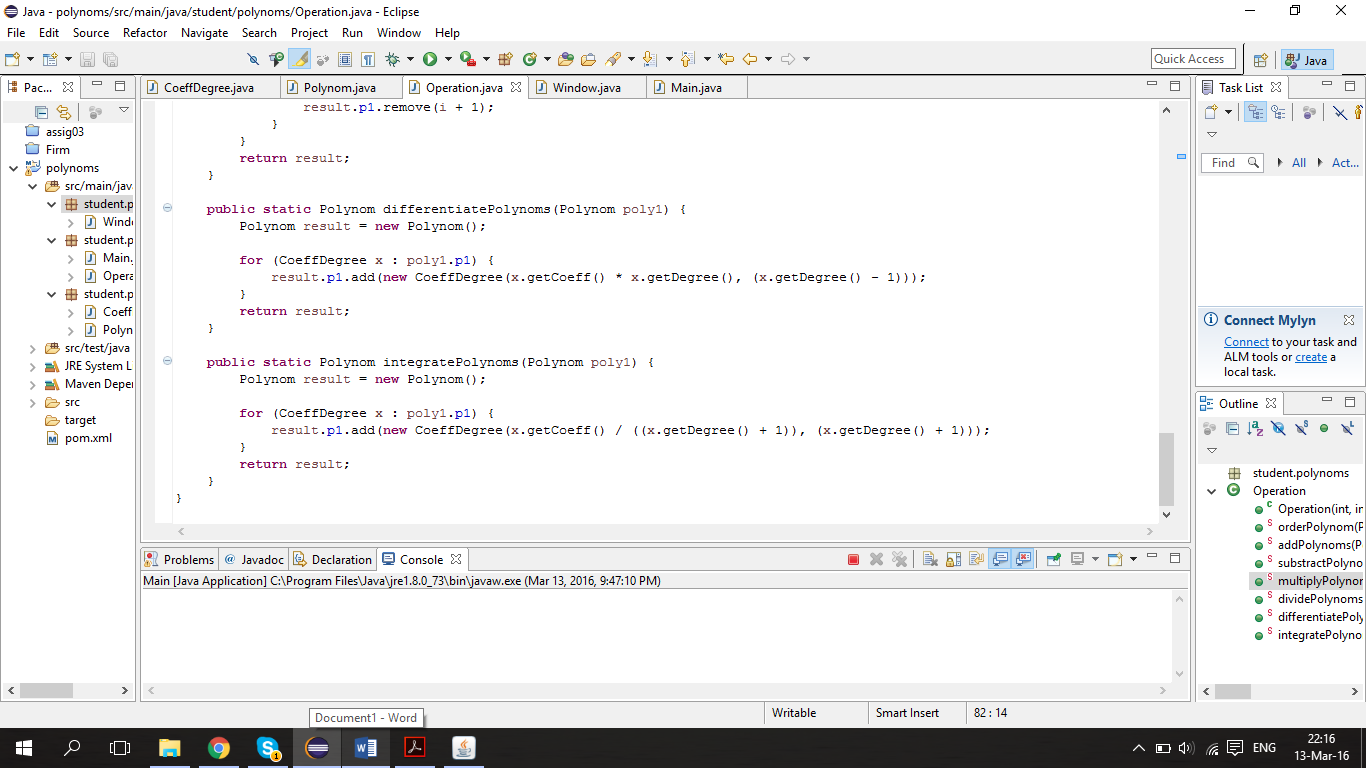


This method implements the basic mathematical operation of differentiation, depicted below:

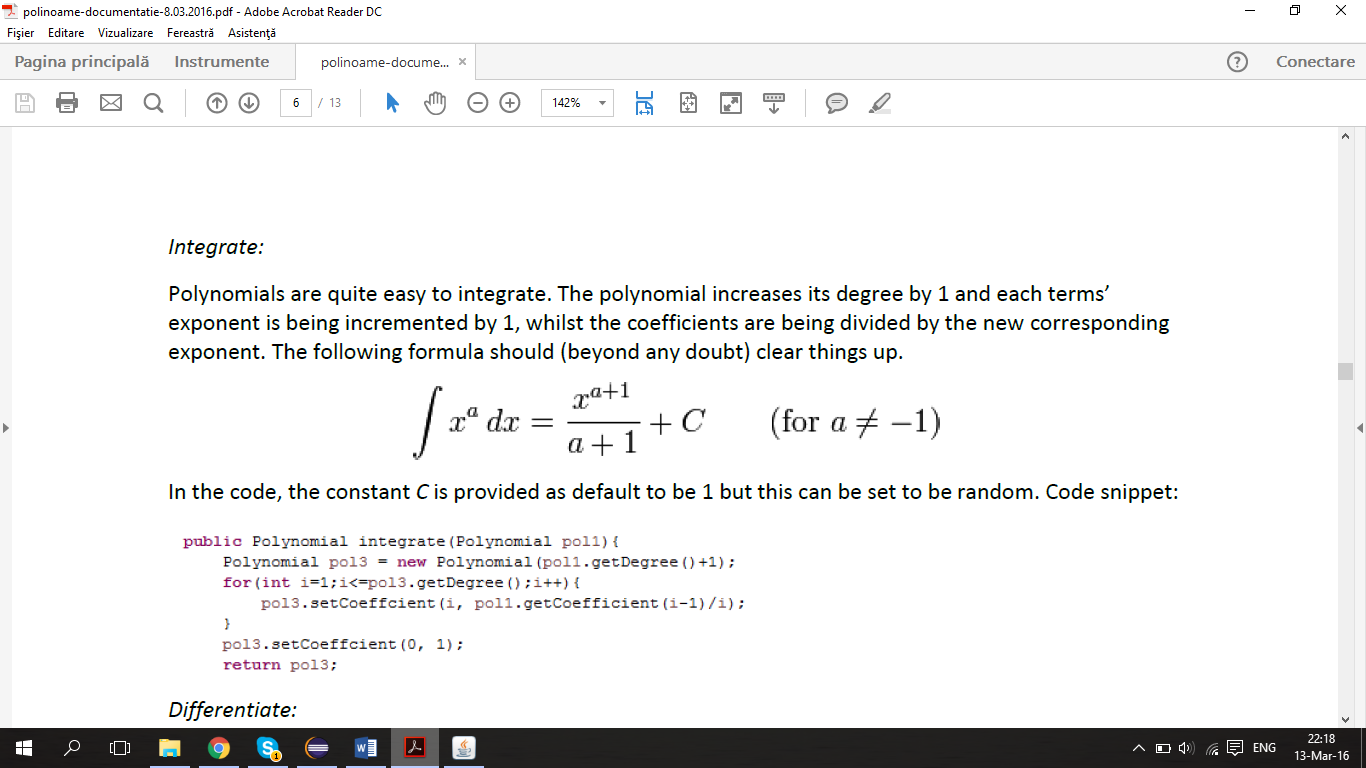


The operation is performed accordingly. The formulae above refers to monomials, however it can me simply extended to polynomials as it is repeated for each monomial element in our polynomial. The way our problem is designed facilitates the ease of implementing this operation

6. Integration



As well as the differentiation mentioned above, the integration method is based on the mathematical formulae, presented next.



The ease of expanding the above formulae, which is aimed for monomials is given by the way the program is modeled. It is simply repeated for each element in the list of monomials, which together form the discussed polynomial.

Class Design

When designing the classes in one project, one should keep in mind as well as focus on what the class is aimed to do and it should that action and only that. As a consequence, the program in discussion is made up of four classes, each serving a different but specific purpose.

The CoeffDegree class, serves the basic need of creating monomials as a base for the polynomials this application is aimed to work with. This class, through the way it is done, facilitates the ease of further implementations and operations. This class contains two attributes and setters and getters for them, which will be used in the development and facilitate access throughout the whole implementation process.

The *Polynom* class serves the purpose of basically creating the polynomials. The constructor of this class calls a text parsing method, implemented by this class. The mentioned text parsing method is used as a conversion tool for the input given by the user. The text box provided by the user interface can only extract string type variables. This could be a potential arising problem for the entire application. However, using this parsing method, we will be able to extract only the desired elements from the given string, mainly and namely the coefficient and degree of each one of the inputted monomials. At first the “*polynom*” will be separated into parts, each containing a format of “coefficient x^degree”. Then we will parse this even further, extracting only the coefficient and the degree, adding the pair to the list of pairs in use, this way creating an internal format polynomial.

The next class created is the Operation class. This class implements the methods mentioned above, all in all supporting the practical goal of our application. All the methods implemented return an additional result polynomial, feature which will be used further in the application.

Lastly, the Window class supports the interface required for the user to operate all the implemented algorithms. Using the basic swing elements we start by creating a new JFrame as a “background”, to which we will add a few elements. The default features for this frame are set, including the size of the frame, its visibility and also the default close operation as well as a give layout, for more appealing to the eye display. We go on by creating the three JTextFields used for input and output purposes. One of the above mentioned JTextField is going to serve the purpose of displaying the result of the operations, as a protection mechanism it will be set as unable to receive modifications. Each one of these JTextField is going to be added to one of the JPanels used to separate the areas of the frame. For the last JPanel we would chose a different aspect. This JPanel is aimed to support the buttons which will indicate the operation performed. Therefore, 6 buttons will be added, each one of them displaying a suggestive character, indicating its purpose. An interesting feature here would be the action listeners applied.

The action listeners implemented serve as the triggers for the result display. Basically, whenever a button is pressed, the strings form the text boxes is being converted and the operation picked performed. The result is displayed accordingly, using an internal format to string conversion method.in the end all the above created elements are added to the frame, resulting the beautifully structured and above presented window, supporting all the functionalities of the program developed.

Packages

The Classes presented above are grouped according to their purpose into 3 packages. The Window class is part of the GUI package as it is responsible for interacting with the user. The Entities package deals with creating and basic functionalities of the entities involved in the project. Keeping this in mind, the *CoeffDegree* and *Polynom* classes belong to the previously mentioned package. Lastly, we mention the general package, to which the other 2 belong, where the Operation class belongs as well as the Main, default class of the project, which will set the execution of the application.

User Interface

Going back to the active interface, the user will be welcomed by a window described by two input text boxes, which will require the user to type in the desired polynomials as strings by the format ”coeff1x^degree1+coeff2x^degree2…”. Then the user has a variety of six buttons from which he can chose what to do with the inputted polynomials. By clicking each one of them a different result will be prompted, according to the decision taken previously. The result will be displayed in a format similar to the input required one, in the text box located below the six buttons aimed for selecting an operation. Prior to testing and making sure the program works as it should, console intermediate results, such as correct parsing of the user given polynomials and results in internal format, were removed.

Implementing and testing

The most used technique of testing was the insertion of the *System.out.println*(…) method in different stages of the development process. Using this method came in handy when the result of a certain operation was needed to be tested for proper working .The main advantage of this method is that it offers quick feedback and it may help locate the unusual behavior source very fast. On the other hand, the fast performance comes with a major disadvantage, its size and many lines of code if it is needed repeatedly and also the impracticality aspect in some cases.

Trial and error was the next step to obtain a perfectly working algorithm. After developing all the required method, any programmer would think that it should work. Unfortunately, this in not always the case. For the first test of proper performing, some input data will be entered, and a proper result should be obtained, of course considering the entered data is of proper type. If the obtained result does not matches the expected one, the first step is to check if the algorithm is correct and make the needed changes and alterations if necessary, which most of the time are.

A good technique and approach is to verify the correctness of the small modules included into the large project. If the correctness and proper performance of the basic parts is ensured, the errors in the final step of the testing are less likely to appear.

If by any chance, the algorithm implemented does not give the desired results, even after many and may modifications, then a logical rethinking must be performed. Taking the problem step by step may give the result through a new approach, or it may help you figure out what is not correct in your first approach. This method may be very effective but it will surely cost more time. Despite that, in some cases, figuring out a new approach takes less time that figuring out the mistake in a poorly implemented and rapidly, in the urge of the moment developed solution.

Results

The most important objectives this project aimed to satisfy have been accomplished. A user friendly interface has been delivered, with its functionalities implemented. All the methods could be further developed and there is also space for improvements in the future. The application offers the basic operations on polynomials which are fully functional and good to go, ensuring the overall success of the project.

Conclusion

The project helped dusting off the polynomials knowledge acquired during the years I have studied in the mathematics and informatics domain. This project also help me get a better understanding of how the initially simple looking but in fact complex operations on polynomials work. It was also a good exercise of design, the way each functionality should be put in the right place, so it could offer the best results. One of the main aspects which I understood from developing this application is the fact that a good initial design and organized initial approach helps a lot when working with the object oriented programming entities. If the base is done correctly and efficiently it saves a lot of time when developing the solution and also excludes some of rethinking processes usually mandatory. Another aspect to be mentioned is that if the initial design of the classes is done properly, the needed calls for the methods and objects can be done in far less words, which is helpful in large, complex projects.

Further Development

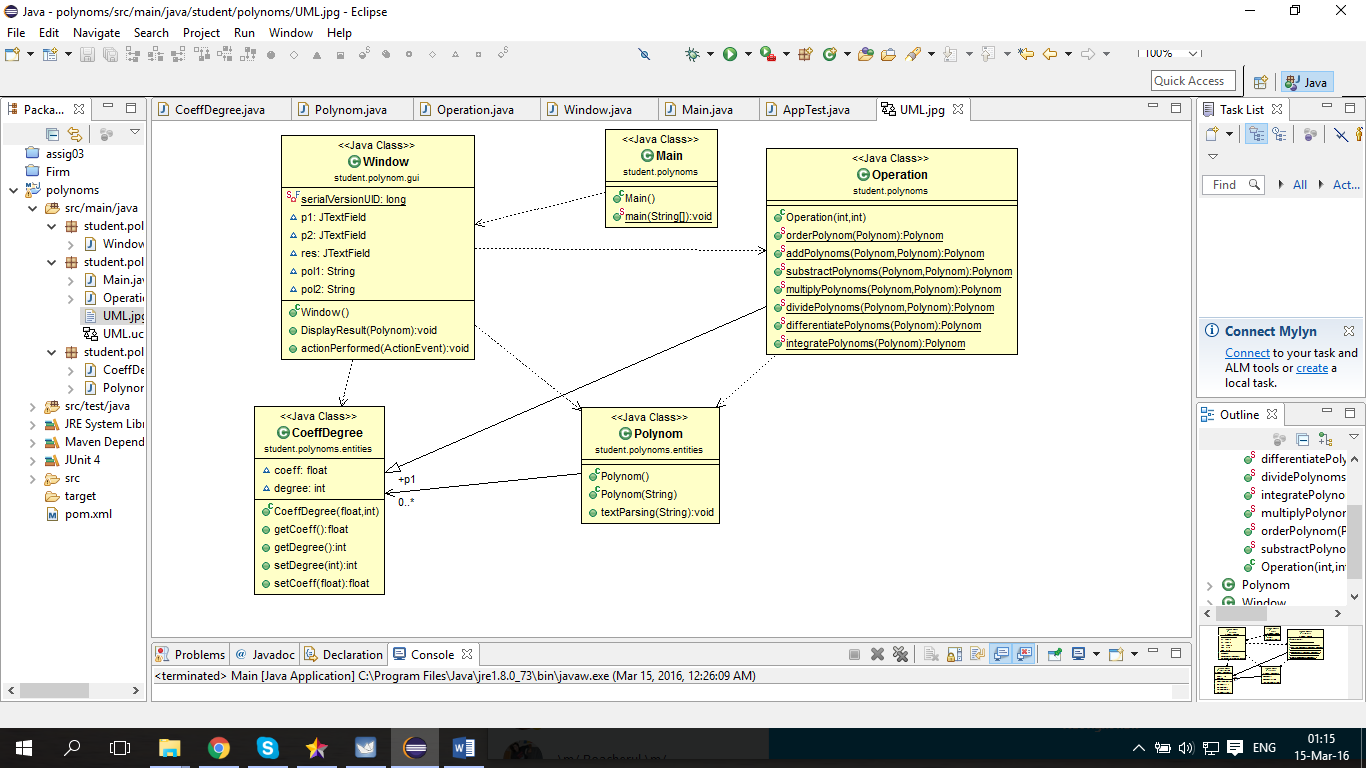
The application presented in this project serves an enormous potential for further implementations of more and more complex aspects .To be noted as future enhancements would be the inclusion of more complex polynomials in terms of coefficients and degrees, meaning larger values, as well as formats. As the user interface interaction window resembles a calculator, the application could be developed to offer full calculator functionalities, for not only monomials but polynomials.as well. The operations offered and implemented could be also further developed, in a manner in which more operation of different complexities could be added. One step forward would be to change the input methods by offering numbers and certain characters responsible for different operations. Another feature could be the selection of the number of polynomials which participate to the operations as well as enhancing that number up to a more complex value. An interesting feature that should be taken into account is the possibility of offering a step by step mathematical solving process of the operation, this way the user being able to understand how the operation was performed, instead of just taking the final result.

As far as the graphical user interface is concerned, if all the above mentioned enhancements are to be implemented they would need a graphical implementation as well. All the extra added operations will require a button for selecting it and if we were to implement a larger number of polynomials participating to the operation, input text boxes need to be added to the final design. If we were to talk strictly about the present application the graphical user interface can be modified so it has a more attractive look. The functionality and ease of use of the present developed application should be enough for the moment, none of them requiring more explicit visualization. One feature that could be eventually mentioned could be the way the input should be typed in, a hint may be offered to the user at the top of the window.

Bibliography:

1. www.stackoverflow.com
2. www.wikipedia.org

UML Diagram



Use Case Diagram

