

Technical University of Cluj-Napoca  
Faculty of Automation and Computer Science  
Computer Science Department

Polynomial Calculator

-JAVA Project-

Coordinating: Claudia Pop

Student: Onaciu Andreea

Group 30414

Table of Contents

[1. Assignment Objectiv 3](#_Toc445796660)

[2. Problem Analysis 3](#_Toc445796661)

[2.1 Modeling 3](#_Toc445796662)

[2.2.1 Model 4](#_Toc445796663)

[2.2.2 View 6](#_Toc445796664)

[2.2.3 Controller Part 7](#_Toc445796665)

[2.2 Scenarios 8](#_Toc445796666)

[2.3 Use Cases 8](#_Toc445796667)

[3.Design (UML Diagram) 10](#_Toc445796668)

[4.Implementation and testing 10](#_Toc445796669)

[5.Results 11](#_Toc445796670)

[6. Conclusions 12](#_Toc445796671)

[7.Future developments of the project 12](#_Toc445796672)

[8.Biography 12](#_Toc445796673)

## Assignment Objectiv

The task of the assignment was:

“ Propose, design and implement a system for polynomial processing. Consider the polynomials of one variable and integer coefficients. ”

The task states that the project must be able to perform some operations regarding polynomial processing, for example:

-addition of 2 polynomials, subtraction, multiplication, division of two polynomials;

-differentiation and integration of a polynomial;

In order to perform the division and integration operations correctly, I used polynomials with real coefficients (double). The project had to have a user graphical interface, which enables users to enter 2 polynomial and perfom the mentioned operations. There will be different buttons for every operation mentioned C The result will be displayed on the bottom of the window. In addition, we had to implement several JUnit classes to test the operations.

## Problem Analysis

## 2.1 Modeling

The process of modeling the problem begins with choosing a design pattern suitable for the requirements of the project. Given the fact that the assignments ask for a graphical user interface, one design pattern that matches these conditions is Model –View – Controller.

* *Model*- The model represents data and the rules that govern access to and updates of this data. In enterprise software, a model often serves as a software approximation of a real-world process.
* *View* - The view renders the contents of a model. It specifies exactly how the model data should be presented. If the model data changes, the view must update its presentation as needed. This can be achieved by using a push model, in which the view registers itself with the model for change notifications, or a pull model, in which the view is responsible for calling the model when it needs to retrieve the most current data.
* *Controller* - The controller translates the user's interactions with the view into actions that the model will perform. In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in an enterprise web application, they appear as GET and POSTHTTP requests. Depending on the context, a controller may also select a new view -- for example, a web page of results -- to present back to the user.

For each of these parts, I created a separate package, to organize and have a clean view of the project. Afterwards, the next step is to focus onto the structure of each part : model, view and controller.

### 2.2.1 Model

Starting with the model, which represents the base of the project, I had to decide how to organize in classes with attributes and capabilities the problem. The following classes construct the model part:

1. Polynomial

The Polynomial class was absolutely necessary as the main goal of the project is to have system capable to perform different operations on polynomials.

Attributes:

Each polynomial has one or more terms, so one attribute of my Polynomial class is an ArrayList of terms, from which a new class was build : Term.

Capabilities:

-besides the constructor, setters and getters, the Polynomial class must be able to provide a string representation (method toString);

-also it can add a term or it can substract a given term (methods : addTerm() and subtractTerm() );

-finding the maximum degree of its term, the polynomial class has the capability to report its degree (method getDegree());

-another method was needed for performing the operations namely getCoefficientOfLeadingTerm;

1. Term

A term is component of a polynomial, that has its own characteristics.

Attributes:

Each term has a coefficient and a degree, features that will help us in performing the operations on the polynomial.

Capabilities:

-for making simpler the toString() method from the Polynomial class, I made a toString() method in this class too;

-besides its constructors, setters and getters, I decided to implement a method to compare 2 terms regarding their degree, so as to be able to sort the ArrayList of terms of the polynomial, for that my Term class implements the Comparable interface, writing its compareTo() method.

The operations that can be a perform on a polynomial take 2 forms : binary operations and unary operations. In a binary operation there are 2 polynomial involved and one polynomial as the result (exception division). In a unary operation, only one polynomial is needed as input and one polynomial gets out as a result.

In order to implement these concepts, I created two abstract classes : BinaryOperation and UnaryOperation.

1. BinaryOperation

The abstract class BinaryOperation contains the abstract method execute(), which takes two polynomials as parameters and return another polynomial. Given the fact that the division operation has as a result two polynomial, the quotient and the rest, I decided that the method execute() to return an ArrayList of polynomials. That means operations like addition, substraction, multiplication will return a list of one polynomial, while division will return a list of two polynomials.

1. UnaryOperation

The class UnaryOperation is an abstract class that has the abstract method execute(), which receives as a parameter one polynomial and return one single polynomial as a results. The operations that enter in this category are differentiation and integration.

1. Addition

The Addition class extends the abstract class BinaryOperation and writes the method execute(Polynomial a,Polynomial b), using the capabilities of the polynomial class: getTerms() and addTerm(). The method will return as first element of an array list, the result of adding two polynomials a and b, which is also a polynomial.

1. Subtraction

The Subtraction class extends the abstract class BinaryOperation and writes the method execute(Polynomial a,Polynomial b), using the capabilities of the polynomial class: getTerms() and substractTerm(). The method will return as first element of an array list, the result of substracting a from b, which gives also a polynomial.

1. Multiplication

The Multiplication class extends the abstract class BinaryOperation and writes the method execute(Polynomial a,Polynomial b), using the capabilities of the polynomial class: getTerms() and addTerm(), and the sort method of the Collections class for sorting the array list of the resulting polynomial. The method will return as first element of an array list, the result of multiplying two polynomials a and b, which is also a polynomial.

1. Division

The Division class extends the abstract class BinaryOperation and writes the method execute(Polynomial a,Polynomial b), using the other binery operations : addition, subtraction and multiplication. The method will return as first element of an array list, the quotient of the division a/b and as second element of the array list, the rest of the division. The method also has the capability to get out from the known error of Division by zero.

1. Integration

This class extends the abstract class UnaryOperation and writes the method execute(Polynomial a). The method will return the result of integrating polynomial a, result which is also a polynomial.

1. Differentiation

This class extends the abstract class UnaryOperation and writes the method execute(Polynomial a). The method will return the result of differentiating the polynomial a, result which is also a polynomial.

### 2.2.2 View

Another part of the project was building a friendly interface for the user to be able to perform the above mentioned mathematical operations on polynomials. This part enters in the view package and was made in the PolynomialFrame class. The class uses Swing and awt classes, to create panels, text fields, buttons. The class extends JFrame, and uses as main panel a GridLayout of 2 lines, which holds in the first line another GridLayout of 2 columns. The first column representing the zone where the users inserts the polynomials. It consists of two text fields, and some labels that help the user understanding how to use the interface. The second column holds another panel with a GridLayout of buttons, the buttons represent the operations that are implemented on the application. When pressing a button, a result of the operation will be displayed in a text field, placed into the second line of the main GridLayout. If the operation is a division the rest will be displayed in another text field.

### 2.2.3 Controller Part

For making a link between the model and the view, the package controller part holds classes that communicate between those two.

1. Converter

The converter class transforms a string intro a polynomial. It has only a method, and it is used when reading the user input and transforming that input into an object of THE class Polynomial.

1. AbstractController

This class holds the main parts on which the problem works on, more precisely objects referring to : the frame, two polynomials, binary operation, unary operation and the converter. This is an abstract class which will be extended by the actual controller.

1. PolynomialCalculatorController

In this class happens the real binding between the model and the view. It contains extends the AbstractController, so it contains references to all important parts of the project. It contains inside classes that implement action listeners to all the buttons on the frame.

- *AdditionButtonActionListener*

*- SubtractionButtonActionListener*

*- MultiplicationButtonActionListener*

*- DivisionButtonActionListener*

*- IntegrationButtonActionListener*

*- DifferentiationButtonActionListener*

All these inner classes, implement the Action listener interface and they write the method actionPerformed() which has to decide what actions to take when a specific button is pressed. In our case the following actions take place: first the method converts the input string representing a polynomial into an object of type polynomial, using the converter object. Then using an object of the corresponding operation perform the method execute(), using the converter results as parameters. The result of the operation is transform into a string using the method toString() of the polynomial class, and after that the string is set into the result text field and in the division case also in the rest text field.

## 2.2 Scenarios

In order to perform the polynomial operations there were multiple scenarios that have to be taken into account for performing mathematically correct the computations.

First of all, because of the fact that the division and also the integration of polynomial can give as a result polynomials with real coefficients, instead of casting the result to integer, I decided to allow polynomial to have real coefficients, in order to have realistic results.

Another problem that had to be resolved was having to deal with the “0” terms, or the null polynomial, more precisely, after performing a computation that gives a result equal with 0, there will be terms in the resulting polynomial with zero coefficient that have to be discarded somehow, and the result that it is showed onto the screen to be “0”, and not “0+0+0”. To deal with this problem I choose to check if the polynomial has only zero terms and show only one term, which means only one zero.

When comes to the division operation, a frequently known error has to be taken into account: Division by zero. To handle this error, the division class which holds the method execute, verifies first if the divisor is different from the null polynomial,

If yes, the division is performed, else the method will throw a Runtime exception, with the message “Division by zero”.

For letting the user known that the division cannot be performed, the division action listener, test itself that the divisor is or not a null polynomial, and based on that performs the division or shows a message ( “Division by zero”) to let the users know that the operation could not been done.

## 2.3 Use Cases

Name: “ calculate ”

Brief Description

The systems waits for the user to enter one, or two polynomial on which to perform different operations. After entering the polynomials, the user should press one of the buttons, depending on which operations he/ she wants to be performed. The result of the operation will automatically be displayed into the result text field.

Actors

The users which want to compute a polynomial operation, they can be students, mathematiciens or any type of person which needs to work with polynomials, and wants to perform polynomial operations in a short amount of time.

Preconditions

At each new run of the application, the text fields will be ready to enter new polynomials by displaying a zero on each text field.

A frame containing labels, buttons and text fields, a controller, two polynomials and a converter, a BinaryOperation and a UnaryOperation objects are already created and ready to be filled up with new data, entered by the user.

The flow:

For a properly use of the application the following facts have to be taken into consideration:

-When entering the polynomial in a string format, it must precisely obey the form from the exemple specified under the text fields, which refers to the fact that the polynomial must have the following form :

C1X^D1+C2X^D2+….CNX^DN

-where C1,..N  are the polynomial coefficients and must be written even if the coefficient is 1; in case it is 0, the term is not mandatory to be written. Another thing that has to be taken into account when talking about coefficients is their sign, if there are negative numbers the sign “-“ should be written without putting any parenthesis, if not the polynomial will not be parse correctly by the method convertStringIntoPolynomial(String polynomial) of the converter class.

-the degrees of the terms D1..N, have to be specified allways, even when there are 0, if not the parser will not interpret the polynomial correctly and no result will be displayed onto the screen.

Only the correct written terms will be taken into consideration when performing the calculus.

After writing in a correct manner the polynomials into the text fields, in the right side the user can choose which operation to be performed on them by pressing the buttons. For a clear view the polynomials are denoted by A and B, and the buttons have the labels: A+B, A-B, A\*B, A/B, Integeral of A and Derivative of A. After pressing one of these buttons a result will be displayed in the text field labeled: “Result”, respectively “Rest” ( in the case of division).

If the user decided to change the operation, he/ she can press other button, without having to write again the polynomials.

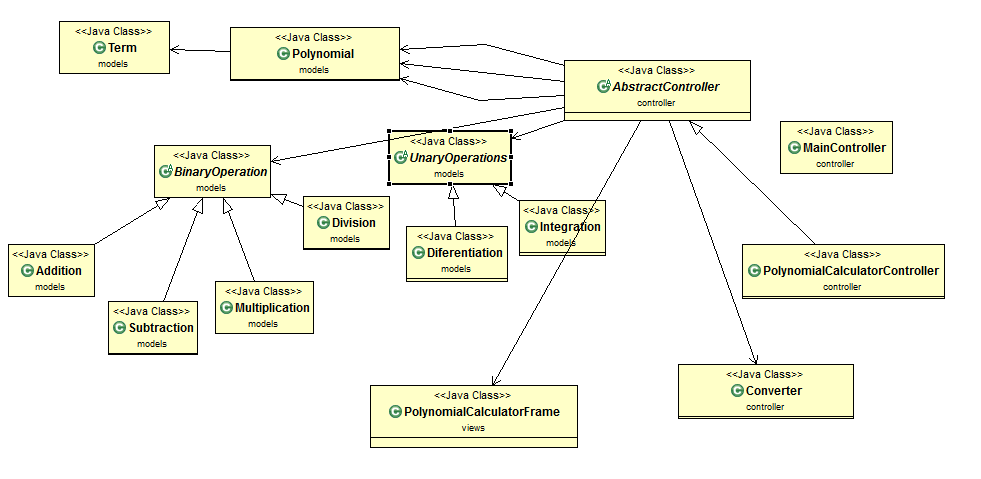
Post Conditions

After the use case is performed, the user enters the polynomials in a correct manner, the polynomial which results after the operation is shown in the text field.

## 3.Design (UML Diagram)

The UML Diagram was build using Eclipse IDE.

This is a more clear view of the diagram, without showing nesting and association multiplicity:



## 4.Implementation and testing

The project was implemented in Eclipse IDE, with the help of its features and some predefined classes and interfaces, the more important are : Collections,Array List, Comparable, Swing, Awt, ActionListener , StringJoiner , regex.Matcher, regex.Pattern .

* The most important class was ArrayList, which came in handy when having to deal with a collection of terms, with a variable dimension.
* Using the Collection and the Comparable interface, I sort the array list of terms.
* The Swing and Awt classes provided the buttons, text field, layouts, labels to create a friendly user interface.
* The method actionPerfomed() form the interface ActionListener helped in making the bound between the view and the model.
* The StringJoiner class is used in the method toString() of the Polynomial class to join the representation of the terms into string all together, so as to form a polynomial string representation. All these where join using the sign “+”.
* The Regex class, was extremely helpful in the conversion of the user input string of the polynomial into a polynomial object. Using the Matcher and the Pattern subclasses, I created a list of terms form of coefficients and degrees. Which where set in a new polynomial object.

The testing part consists of 6 JUnit test classes, reffering to the operations that can be perform on the polynomials :

* AdditionTest
* SubtractionTest
* MultiplicationTest
* DivisionTest
* IntegrationTest
* DiferentiationTest

The methods which were tested are :

* Method execute(Polynomial operand1,Polynomial operand2) from Addition class;
* Method execute(Polynomial operand1,Polynomial operand2) from Subtraction class
* Method execute(Polynomial operand1,Polynomial operand2) from Multiplication class
* Method execute(Polynomial operand1,Polynomial operand2) from Division class
* Method execute(Polynomial operand1) from Integration class
* Method execute(Polynomial operand1) from Differentiation class

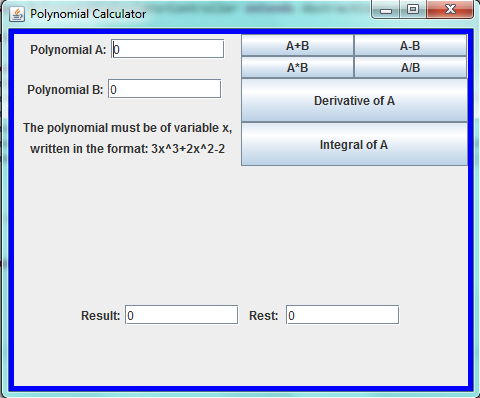
The test where perform by trying simple example and check in if the result of the methods which where tested yields with the correct results.

## 5.Results

The results obtained can be viewed and tested in the user interface.

To verify if the application perfoms well the operation, one can perform the operations by hand on paper and check if they reseamble with the actual response of the application.

Here are some print screens which show the design of the user interface :



## 6. Conclusions

The assignment help me improve my knowledge regarding Object- Oriented- Programming. I learned how to design from a project, how to transform some mathematical operations and concepts into classes and methods, that satisifies the object-oriented – programming rules and concepts. I learned how to test my program and the advanteges of writing test cases. One of the major advantage is the amount of time saved during the actual implementation and the fact that the tests assure partial correctness of my algorithms. By using JUnit tests its easier to verify your project, without having the need to type every time and input, compute the output and verify if the results matches. The Java Eclipse Ide, provides use a simple mechanism that lets you see that your tests done well by showing a green rectangle, or if not a red one.

## 7.Future developments of the project

1. The project can be improved by adding more operations to the Polynomial Calculator, for example: evaluating the polynomial, calculating its roots, integral at some specified point and others.
2. Another idea will be to improve the way the polynomial it is display on the result.
3. To have messages for the user to explain what is wrong, when he/she enters a wrong input.
4. To improve the user interface, by making it more friendly, and more easy to use.

## 8.Biography

**Books**:

- Joshua Bloch, Effective Java (2nd Edition);

- [Kathy Sierra, Bert Bates](http://shop.oreilly.com/product/9780596009205.do#tab_04_2), Head First Java (2nd Edition), O'Reilly Media;

- Barry Burd, Java for Dummies (5th Edition), Wiley;

**Websites**:

- <http://stackoverflow.com/>

- <https://www.oracle.com/java/>

-http://www.oracle.com/technetwork/articles/javase/index-142890.html