Technical University of Cluj- Napoca

Faculty of Automation and Computer Science

Programming Techniques

Homework Assignment 1

Student: Eric-Andrei Dumea

Group: 30423

Supervising Teacher: Cristina Bianca Pop PhD.

# Table of contents

1 . Problem specification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2 . Problem analysis, modelling, scenarios, use-cases . . . . . . . . .3

2.1 Problem analysis, modelling . . . . . . . . . . . . . . . . . . . . . 3

2.2 Scenarios, use cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

3 . Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.1 Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.2 Classes Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.3 Packages & Interfaces . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

3.4 User Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

4. Implementation & testing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

5. Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

6. Conclusions, further development, self-taught knowledge . . . . . . . . . . 10

7. Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

# Problem specification

Propose, design and implement a system for polynomial processing. Consider the polynomials of one variable and integer coefficients. The application will be able to perform the following operations:

* Addition of two polynomials
* Substraction of two polynomials
* Multiplication of two polynomials
* Division of two polynomials
* Derivation of a polynomial
* Integration of a polynomial

# Problem analysis, modelling, scenarios, use cases

## Problem analysis, modelling

For this application I used the paradigms of Object Oriented Programming in Java. Object-oriented programming (OOP) refers to a type of computer programming (software design) in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure. In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects. The main concepts of Object Oriented Programming are Encapsulation, Abstraction, Inheritance and Polymorphism.

*Inheritance* provides a powerful and natural mechanism for organizing and structuring your software. Classes inherit state and behavior from their superclasses, and you can derive one class from another using the simple syntax provided by the Java programming language.

*An interface* is a contract between a class and the outside world. When a class implements an interface, it promises to provide the behavior published by that interface.

*A class* is a blueprint or prototype from which objects are created.

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class.

An application solving such operations could be of use to learning students in the field of mathemathics, such as high school students, which need a way to know whether their homework is correct or not, before handing it in. Also, another application for such a system would be for making engineers’ life easier, because one can find polynomials everywhere in this domain, and a quick way to make operations could save a lot of time. The user-friendly Graphical Unit Interface (GUI) makes it easier for people that aren’t familiarized with mathematical -oriented programming languages to introduce the polynomials and to compute the desired output.

A polynomial is an expression that can be built from constants and symbols called indeterminates or variables by means of addition, multiplication and exponentiation to a non-negative power. Two such expressions that may be transformed, one to the other, by applying the usual properties of commutativity, associativity and distributivity of addition and multiplication are considered as defining the same polynomial. Polynomials appear in a wide variety of areas of mathematics and science as they are used to form polynomial equations, which encode a wide range of problems and to define polynomial functions, which appear in settings ranging from basic chemistry and physics to economics.

Each polynomial consists of monomials. In mathematics, a monomial is, roughly speaking, a polynomial which has only one term.

Based on the above, it is obvious that, the choice of main entity was the polynomial. The project was built around the concept of polynomial, divided into monomials. Apart from these two classes, the project contains a class for the graphical user interface and one for the operations that can be made on the polynomials.

## Scenarios, use cases

The program will have a basic interface with buttons and some textboxes. One must provide a valid input, such a polynomial is written as 3x^2+1x^1+5x^0. After that, the user should press the button associated with the desired operation, and the result should appear in the third textbox. Also, the user is provided an exit button, an information button and a clear button. We have to mention that integration and derivation are only available for the polynomials introduced in the first textbox.

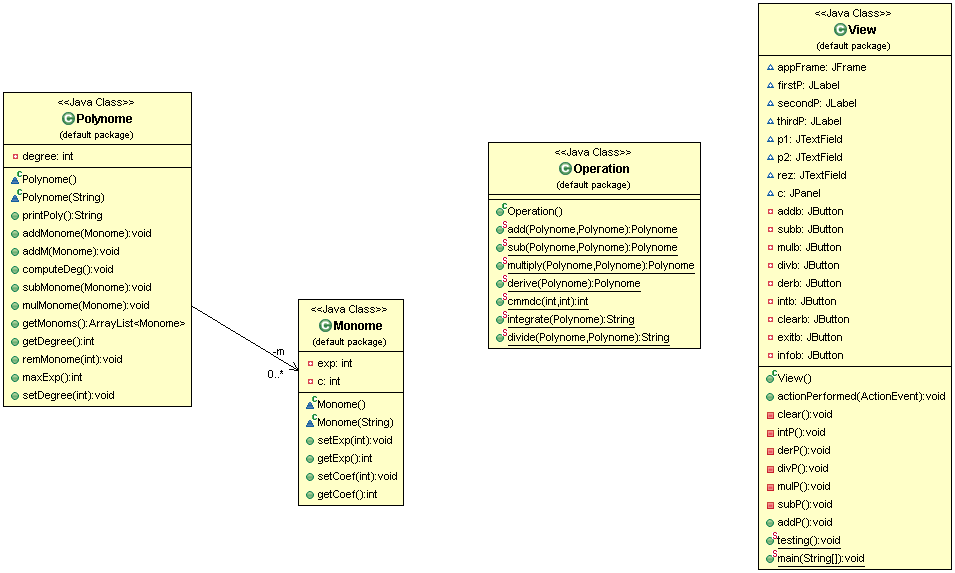
As an example, the user has entered two simple polynomials.

12x^4-6x^3+2x^1+7x^0 in the first textbox and 3x^2+2x^0 in the second. The results will be, after:

* Addition: 12x^4-6x^3+3x^2+2x^1+9x^0
* Substraction: 12x^4-6x^3-3x^2+2x^1+5x^0
* Multiplication: 36x^6-18x^5+24x^4-6x^3+21x^2+4x^1+14x^0
* Division: Quotient:4x^2-2x^1-2x^0 Remainder:-2x^2+6x^1+11x^0
* Derivation: 48x^3-18x^2+2x^0
* Integration: 12/5x^5-3/2x^4+1x^2+7x^1

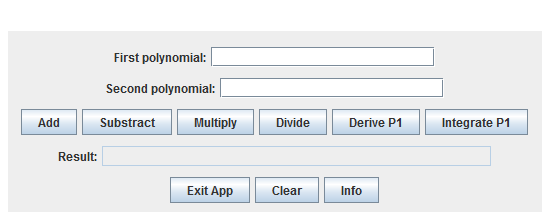
# 3.Design

## 3.1 UML Class Diagram



## 3.2 Classes design

1.View Class

The “View” class implements the interaction with the user, also known as GUI or Graphical Interface Unit. The window that pops up at runtime:

Also, the View class contains the main method, *public static void main(String[] args),* which creates a new instance of the View class, thus calling the constructor, followed by the *v.appFrame.setVisible(true)* instruction, making the GUI visible. The constructor builds up the GUI part, using the javax.swing framework. Here, we have a JFrame called *appFrame* that is actually the window that pops on the screen, the canvas for the other visual components. Also, I used a BoxLayout layout JPanel as a container for the other components. I divided the components into blocks ( divide et impera ), assembling them in the container. I used 6 JPanels using FlowLayout to achieve this.

The View class implements the *ActionListener* interface, thus it contains the *public void actionPerformed(ActionEvent e)* method. This method specifies the behaviour of the application at different stimuli (e.g. – Add button has been pressed). Because of this, I’ve made a method for each button pressed (e.g. divP, addP, subP..).

Also, this class contains developer’s information, in the form of the *testing()* method. It contains hardcoded strings of polynomials that helped at the debugging.

2. Monome class

This class is the nucleus of the application, because every Polynome will be splitted in monomials. We have only two fields, **private** **int** exp **private** **int** c; which contain the exponent and the coefficient of the monomial. We have two constructors for this class, one without parameters, designed specifically for the auxiliaries needed in development, and one with a String parameter. The last one, basically splits the coefficient from the exponent. The String fed to this constructor looks something like cx^e, where c and e are integers, representing the coefficient and the exponent of the monomial. Basically, it’s just a string parsing algorithm, where the coefficient and the exponent are split (using *split(delim)* method).

Also, there was a need of setters and getters for both fields, as they are required in every single operation.

1. Polynome class

If the Monome class is the nucleus of the project, the Polynome class is the atom of this application. It contains two fields: the degree of the polynomial (the max coefficient of the monomial list) and the list of the monomials, using the List interface implemented by ArrayList. I needed a getter for this list, and a *computeDegree()* method, plus the setter and getter for the degree field.

As in the Monome class, there are two constructors here too. One without parameters that sets the degree of the polynomial 0, and the more elaborate one which builds the polynomial out of the String fed to it. First of all, the minuses are replaced with “+-“ such that, when doing the String parse, negative numbers won’t be truncated, or made positive.

Also, I used an inline-comparator definition for sorting the arraylist by exponent, to avoid declaring another class that implements Comparator<Monome>.

Another method in this class is *printPoly()* which, basically, does the opposite of the constructor. This function generates a string in the format recognized by the constructor, which can eventually be printed in a textbox.

The *addMonome, mulMonome, subMonome* are auxiliary methods, needed in the Polynomial class to do the operations. The first one checks whether the monomial to be added exists (actually, the monomial with the same exponent), and if yes, adds only the coefficient, if no, it adds the monomial to the ArrayList. The subMonome method works basically the same, just it substracts the coefficients, or adds the monomial to the Arraylist, with the sign -. The mulMonome method multiplies every monomial existent in the ArrayList with the one fed as the parameter of it.

1. Operation class

This class contains the methods to do operations on polynomials, hence static methods with Polynomial arguments, except for one.

Addition is done in the add method, with arguments two Polynomials, and makes use of the addMonome method in the Polynome class.

Also, there is a method for every operation that works in a similar manner to this one.

The cmmdc() method is a helper method for the integration operation. As the description for this project states: the application is for integer-coefficient polynomials. Also, my focus was to do this application student-oriented. Thus, the integration and division methods are designed specifically to fulfill a 12-grade student’s needs in terms of polynomial calculus. This is also why the division and integration methods return Strings. The cmmdc ( great common divisor in English – GCD) is used to simplify the fractions that result from integration.

## 3.3 Packages and interfaces

A Java package is a mechanism for organizing Java classes into namespaces. Java packages can be stored in compressed files called JAR files, allowing classes to download faster as a group rather than one at a time. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality.

In this application, I used:

* javax.swing.\* , java.awt.\* and java.util.\* packages mainly for the GUI part and operations on ArrayLists
* org.junit.\* used for testing

Interfaces used: ActionListener implemented by the GUI class (View).

## 3.4 User Interface

As I’ve previously mentioned, the user interface is pretty much basic and easily-understood, even by nonfamiliarized people. When running the application, a window will open and it will provide to the user the possibility of giving inputs and choosing what operation he likes to be executed. This window is constructed in the GUI class using some predefined classes and instructions.

We use buttons for every method, three JTextFields , two for the input and one for the output, and labels to make things easy. Also, some auxiliary buttons are used, such as “exit” - exit the application, “clear” - to clear the text boxes and “info” – an Message Box pops up, with instructions to use the app.

# 4. Implementation and testing

## Implementation

As an input condition, the polynomial entered in the first two textboxes should be of the format:

[-] + coefficient + “x “+ “^” + power

Where: The minus is optional, also there is no need for + when the first term of the polynomial is positive. The app works only with x as a variable.

If there are some coefficients zero, the app will work fine, but preferably, for the sake of memory managing, the user is advised not to enter polynomials with terms that have coefficient=0. Before the user proceeds to the next step, he should check once more if the polynomials were introduced in the specified format.

After the polynomials were (correctly) typed in the two upper textboxes, the user should choose the operation. This can be done by simply pressing the button of the operation of choice. As I’ve previously said, for now, the derive and integrate operations only work on the polynomial inserted in the first textbox. For further versions, this issue will be fixed.

Initially, for testing and debugging purposes, a method with no parameters was declared in the View class, named symbolically “testing”. It contains hardcoded input for the operations, and it allowed the developer to purely understand what he’s doing.

For a better testing, a Junit Test Case was created for verifying the proper working of each of the classes. A JUnit test is a method contained in a class which is only used for testing. This is called a Test class. This method executes the code under test. We use an assert method, provided by JUnit or another assert framework, to check an expected result versus the actual result. These method calls are typically called asserts or assert statements. We should provide meaningful messages in assert statements. That makes it easier for the user to identify and fix the problem. This is especially true if someone looks at the problem, who did not write the code under test or the test code.

One operation that raised some problems in the design stage was the division. The problem specification said that “integer polynomials” are used, so I supposed that, this way, I can also make the application more student-friendly, as I’ve been struggling in high school to find an integer-dividing-calculator, as my teacher told me. Another problem raised by the division algorithm was the result. Two polynomials are returned: the quotient and the remainder. An easy solution for this has been to write the method in such way, it returned a string that already concatenates the quotient and the remainder.

Another operation that raised difficulties in the design stage, but mainly in the development stage was the integration. Again, for the same reasons as in the division part, I chose to work with integer coefficients. This time only, there were fractions needed. I used the great common divisor (GCD – cmmdc) algorithm to simplify the fractions that resulted during the integration. Also, this is again a method that returns a string, because the main Polynomial print function *( printpoly( ))* isn’t capable of dealing with this format. The string building is provided inside this method.

# 5. Results

The user interface provides, even to the users with little knowledge in this subject, the possibility of performing polynomial operations due to the fact it is easy and very simple to use. The results are displayed in an easy-to-understand form, and the Clear button add some flexibility to this application. Also, the info button enables the user to have a quick-access mini-manual at the tip of his mouse pointer.

# 6. Conclusions, further development, gained knowledge

In the end, some things were easy to implement, like the addition or the substraction, but there were some problems with them at the beginning too. The division and integration took most of time allocated to the project because they seemed easy on paper, but when I wanted to transcribe them into code, problems arose.

As a further development, the first thing that should be a better exception system, especially at the input. Also, some kind of slider that toggles between engineers and students, or more specific, between double and integer coefficients could be implemented. More functionality, like Fourier/ Laplace transformations could also be added.

This project/assignment taught me, as a first thing, to do a proper GUI using code only, to use Junit and to do a proper documentation (as in following a structure and using English extensively). Also, the main benefit of this project was the refresh on the Java coding skills.

# 7. Bibliography

* <http://coned.utcluj.ro/~salomie/PT_Lic/3_Lab/HW1_Tema1/Tema1_HW1_Barem.pdf>
* <https://en.wikipedia.org/wiki/Polynomial>
* <http://users.utcluj.ro/~crisb_pop/PT2017_L1.pdf>
* <https://en.wikipedia.org/wiki/Monomial>
* <https://en.wikipedia.org/wiki/Polynomial_long_division>
* <https://stackoverflow.com> – for some quick answers to basic questions
* <http://users.utcluj.ro/~jim/OOPE> and the moodle Lab session guides of Mr. Joldos.
* <https://docs.oracle.com/javase/tutorial/java/concepts/>