Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

2nd Semester 2016-2017

Polynomial Processing System

Programming Techniques

Homework 1

Student Ieremiasș Viorel

Group 30422

Contents

[1. Abstract 3](#_Toc477242907)

[2. Problem Specification 3](#_Toc477242908)

[3. Project Objective 3](#_Toc477242909)

[4. Problem analysis 4](#_Toc477242910)

[4.1. Modeling 4](#_Toc477242911)

[4.2. Use cases 4](#_Toc477242912)

[5. Design & implementation 5](#_Toc477242913)

[5.1. Top level design 5](#_Toc477242914)

[5.2. Class design 8](#_Toc477242915)

[6. Testing 12](#_Toc477242916)

[7. Conclusion 15](#_Toc477242917)

[8. Further development 15](#_Toc477242918)

[9. Bibliography 15](#_Toc477242919)

# Abstract

Polynomial calculations are very often encountered in the work of a high school student, college student or professional in any field of engineering. That is because polynomials offer a great versatility, can represent mathematically almost any process, are simpler than other functions and aren’t computationally intensive for computers, which cannot represent more advanced functions like exponentials, logarithms and others directly.

Although the algorithms for the basic operations with polynomials are not intellectually demanding, the amount of repetitive effort that is put in even the simplest of the operations is a major source of inefficiency when working directly, “with pen and paper”, with polynomials.

Thus, the development of an easy-to-use, intuitive, light and user-focused application for processing and working with polynomials is an idea worth following. Using only Java for both the user interface and background programming, one is able to develop a standalone application which includes the complete set of basic polynomial operations, aimed to shorten the time and effort of the user.

# Problem Specification

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

# Project Objective

The objective of the homework is to analyze the problem of polynomial processing systems, to design such a system, program it, along with a user interface and test the application.

# Problem analysis

## Modeling

In order for the application to fullfill the requirements of a complete polynomial processing system for the basic operations, it must be able to perform addition, subtraction, multiplication and division, differentiation, integration, value evaluation at one point, root finding and equality check. The first 6 of the operations are implemented in the present version, while the last 3 are under work.

At a quick analysis of what polynomials are and how they are operated with, it became obvious that polynomials can be processed using an OO approach and that such an approach not only obeys the OO fundamental principles, but takes great advantage of them, in a clean, simple implementation that highlights the concepts at the base of polynomials in the most pure form, complying with the mathematical principles too.

The central principle when modeling the problem was to break up the polynomial in its constituent elements, the monomials, the simple building blocks, represented by 2 important numbers, the coefficient and the degree.

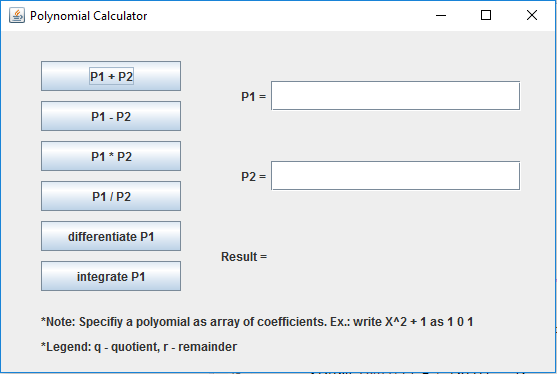
This approach led to the need to implement the same set of operations at the level of monomials, and this may seem redundant and unefficient. But the great advantage came in the fact that, as a consequence, the operations at the polynomial level were greatly simplified, reduced to simply determining how the terms of the 2 polynomials are to be combined together and what operations are to be applied on them. How those operations are implemented was already establised at the previous level and wasn’t a problem anymore.

The result is a model in which responsability and algorithmoc effort is distributed on two levels, monomial and polynomial. With such an approach, not only errors can be detected and isolated easier, but the model can be more easily changed if new features are to be added later on, by simply adding the operation at two levels of the design.

## Use cases

The communication with the user is realized by means of a graphical user interface (GUI) pictured below. It has a minimalist design and is aimed to provide a clean look, be easy to work with, easy to understand at first use, in a window that doesn’t occupy too much space on the screen, while keeping approrpiate dimensions for the buttons and the other fields.

Instructions at the bottom of the window provide short, but clear indications as to how to input data into the application and how to interpret obtained results.



If however data in the input fields is not correct, the user is kindly warned that a mistake sliped in. A warning text appears in the proximity of the input area where there is a problem, so the user is also informed exactly where he the data he entered is not in the correct format.

This makes the application easy to use even for unexperienced users.

# Design & implementation

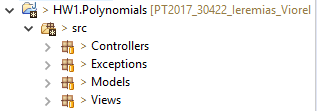
## 5.1. Top level design

At the top level, the project is structured based on the Model-View-Controller (MVC) design. It divides a given application into three interconnected parts in order to separate internal representations of information from the ways that information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development. which presents multiple advantages, like high cohesion, low coupling, and ease of modification.

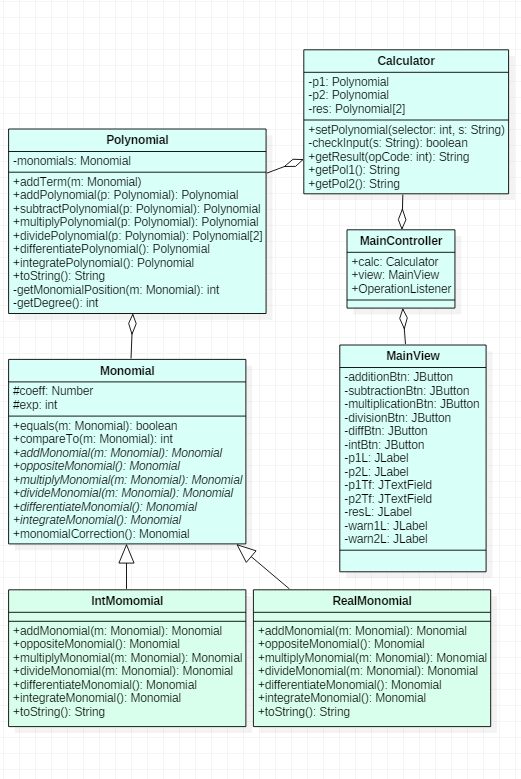
The model is the central component of the pattern. It expresses the application's behavior in terms of the problem domain, independent of the user interface. It directly manages the data, logic and rules of the application. In this project this is realized by a collection fo classes that model the problem of working with polynimials on multiple levels.

The view consists of one class that builds the window on which the user operates. As mentioned before, it was designed to be wasy to use and understand and not to crowd the working space of the user.

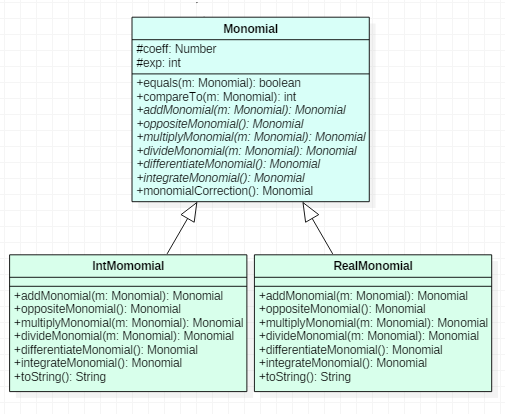
The controller has the purpose to manage the resources of the model and of the view. However, it does not execute anything, it only transmits messages between the two and commands their actions by means of messages. Action listeners are very imoprtant entities that respond to changes of the interface and command specific actions. A project that is dependent when producing the output on the input of the user at one time is an event-driven project and action listeners are an essential part in this case of architectures.

These components of the project also correspond to the main packages in which it is structured. There is a package dedicated to the models that shape the entities, one to the classes that implement the view, one for the controller and finally one package that groups together the user defined exceptions that might occur during the usage of this application.

As seen in the class diagram, pictured below, there is not a strong coupling between the components of the project, which means that the code is easier to maintain, to modify and update, to debug and has the property of interoperability.

Project class diagram

## 5.2. Class design

**Monomial, IntMonomial & RealMonomial classes**

The Monomial class models the basic block of a polynomial, the monomial. This class has 2 fields, one for the coefficient and one for the exponent (degree). This class’s capabilities represent the basic polynomial operations, applied at the level of individual monomials. One monomial has the capability to participate in an operation with another Monomial object and produce a result of the same type. However, the operations are not actually implemented directly in Monomial.

To create a distiction between polynomials with integer and with real coefficients, the Monomial class is an abstract one, the field ‘coeff’ is a Number object and the methods are abstract too. It is extended by 2 subclasses, IntMonomial and RealMonomial, which have constructors that receive an Integer, respectively Double object and implement the operations accordingly. The consequence is that, if one of the terms that participates to an operation is of type real, the result is also real, otherwise it is a monomial with integer coefficient. In the RealMonomial class this is straightforward action that does not require any additional effort, since the object that calls the operations and consequently at least one of the operands, is of type real. However, if the operation is called from an IntMonomial object, the type of the result depends on the type of the other operand. Determining type of the second operand tells what is the correct type of the result and this is achieved by using the instanceof operator on the other operand.

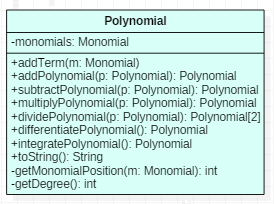
The different implementations of the addMonomial, oppositeMonomial, multiplyMomonial and divideMonomial in the 2 subclasses take advantage of overridding and polymorphic behaivor.

Since, when comparing monomials for the purpose of operating with them, the field that is used as reference is the exponent, in the Monomial class, the equals() and compareTo() methods were rewritten. Equality of two monomials is based solely on the equality of the degrees, and the relation of order is modeled in such a way, that sorting a list of Monomial object after the natural order of the class, the objects with higher degree come earlier in the list.

This approach also serves a more important purpose higher in the hierarchy of the project: a list of monomials can be sorted with the sort() methods in *Collections*, according to the new natural order, without the need of an artificial comparator.

The *monomialCorrection()* method is a special one and it’s purpose is to repair the minuscule errors that can appear in the representation of real numbers. If after an integration for example, the result is actually an integer stored as a real number, its representation might include residual decimals that are due to calculus errors that propagated during the calculations. If this is the case, if the real number is very close to either its floor or its ceil, then the RealMonomial object is replaced with an IntMonomial one.

**Polynomial class**

The Polynomial class contains a single attribuite, represented by a LinkedList of Monomial objects, called *monomials*. A linked list (LinkedList) was chosen over a dinamically allocated array (ArrayList), because there is little to no need for indexed accessing of elements (there is no link between a monomial’s position in the list and its degree) and because all the other operations are more efficient.

One important constructor for this class is represented by the parameterized constructor that receives a static array of numbers. From this, a polynomial is built, term by term, on the rule that the position in the array received as input determines the degree of the term. This constructor is vital when constructing the polynomials from user input.

One of the most important methods is *void addTerm(Monomial m)*. The implementations of this method verifies if there already is a monomial of same degree as m in the polynomial and adds them together by calling the *addMonomial()* method in Monomial class. Otherwise it simply adds the term to the list of monomials.

As a consequence, addition and subtraction of polynomials is done by simply adding the terms of the 2 polynomials to the result, with the mention that in the case of subtraction, the opposite for each term of the second polynomial is added. Multiplication is equally simple, just adding to the result the monomials that result by multiplying each term of one polynomial to each term of the other polynomial. Differentiation and integration are also term-wise operations, that are completed in one traversal of the list of monomials.

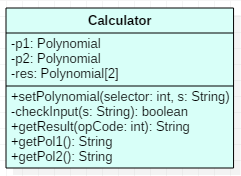
Division is implemented iteratively, by finding a new term of the quotient at each iteration, until the degree of the last dividend is smaller than the degree of the divisor. The remainder is obtained by subtracting from the original dividend the product of the divisor and of the quotient.

The methods *getDegree()* and *getMonomialPosition()* are auxiliary and are used inside the methods that implement the operations, thus they are private to the class. They provide the degree of a polynomial, the greastest of the exponents of the terms, respectively, the position of a term of a specific degree inside the polynomial. There is also an overridden version of the toString() method that is used to prepare the poynomial for displaying in the GUI.

A few important observations about the way the operations on polynomials are implemented:

* The implementation of the operations on individual monomials assures the fact that in the Polynomial class, it is only necessary specify how a certain operation combines the monomials of the two polynomials, and not what is the result of such a combination;
* this feature enforces a clear distinction between monomial and polynomial and promotes abstraction and encapsulation: an object of type Monomial knows how to relate or operate with other monomials, and at the Polynomial class level, this is not visible and doesn’t need to be; a polynomial only needs to know how to relate with another polynomial
* great versatility – as mentioned above, inside the polynomial, there is no need to know how monomials are operated with; this means that, as long as is has the methods of addition, subtraction and so on, any object could be suppiled as Monomial, in the architecture; this opens up the door to working with polynomials made up of all kinds of entities, not only numbers, i.e. symbolic calculus
* at the level of Monomial class, and it’s 2 subclasses, this design makes use of inheritance and polymorphism too; although IntMonomial and DoubleMonomial share the same concept and the same fields, they implement the operations differently in the sense that they output result accordingly to the type of the monomials that participate as operands; overridding the methods of Monomial means that, even though in the Polynomial class, all monomials are seen as Monomial objects, despite they come in 2 flavors, at run-time, dynamic binding assures that the specific method is chosen, for the type of object that is represented by that reference;

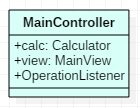
**Calculator class**

This class represents the top class of the Model. It has as attributes two polynomials p1 and p2 representing the operands and an array of length 2 of polynomials representing the result (the second polynomial in the result is used only in the case of division, as remainder).

The method *setPolynomial(int selector, String s)* is concerned with building the polynomial specified by *selector* starting from the string *s.* In case the input is not in the correct form, this method that throw two types of user specified exceptions, *WrongDataFormatException* and *NoUserInputException.* This exceptions, if thrown, are handled in the MainController.

The other important method *getResult(int opCode)* computes the result of the operations specified by the parameter opCode, stores it in the *res* field and returns it as a String object.

**MainController class**

Besides having as attributes a Calculator entity and a MainView one, the controller also has an inner class member, OerationListener, which extends the ActionListener class.

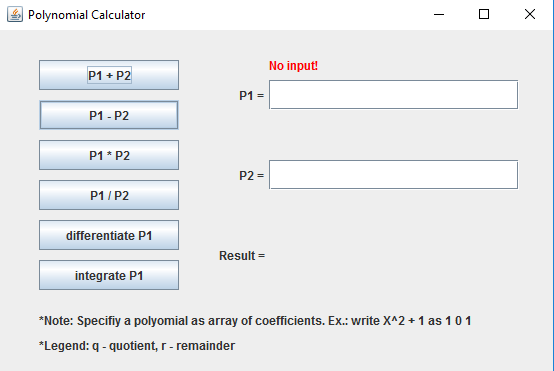
The class is needed to detect the modifications of the interface, done by the user, and act accordingly. In our case, we would need to detect actions performed on the buttons for the addition, subtraction and all the other operations, and compute the result with the polynomials on the input fields. To avoid defining 6 different listeners, the implementation is based on a more efficient solution, that adds a new field to the listener, called *opCode.* This integer determines what operation is to be performed when the button is pressed. And although each button was assigned the same class of listener, that is *OperationListener*, it was with a different opCode, according to a convention established in the code.

**User-defined exceptions**

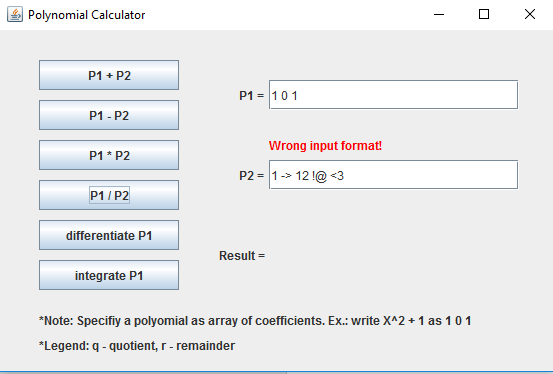
Exceptions are important to identify and handle errors and other exceptional events. Besides the exceptions from the Java library that can be thrown, 2 other situations when inappropriate use of the application can generate errors were identified. First of is when the user tries to obtain a result when there is no data in the input text fields. This situation throws a *NoUserDataException.* If however there is input in the 2 fields, but it is not in the correct format, a *WrongDataFormatException* is thrown. Both are thrown in the Calculator class, if it is the case and are handled in the MainController.

# Testing

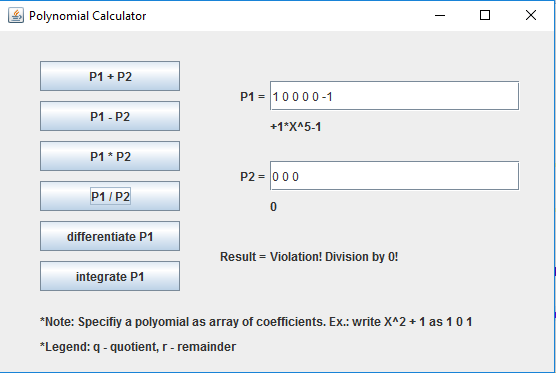
The application was subjected to tests that covered multiple use cases and handled each situation exactly in the way it was designed to do. Let’s consider the following corner cases, that might cause problem:

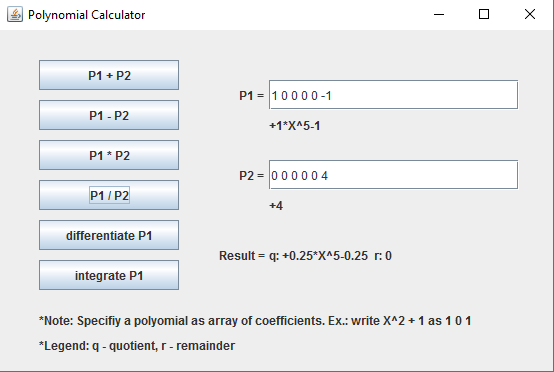


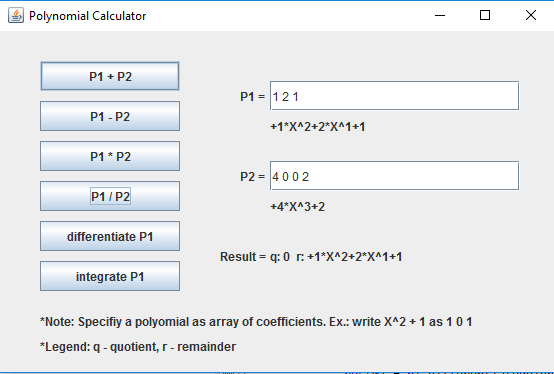
* Trying to obtain a result, with no input: in this case the application warn the use that it cannot perform any operation since it is not given any operands. A red warning message appears above the text field where there is a problem



* Inserting any characters that are not numbers, space or minus sign in the correct format. Again, a red warning text informs the user about the nature of the exceptions and where it occured.

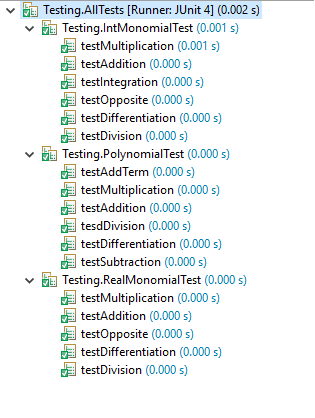


* Division by zero: Another important exception occurs when trying to divide by a null polynomial. In this case, in the result area, there is a message informing about the violation that took place.
* Division with a scalar: when the polynomial P2, the divisor in the case of the division operation, is a scalar, the result is actually the polynomial P2, divided by that constant



* Divisor has a higher degree than the dividend: in this case, the quotient is simply 0, while the remainder is equal to the dividend.

It is important to notice that, except the division by 0 case, all the other special cases for the division are not artificial, but represent the normal functioning of the algorithm.

Besides than that, testing was carried out using the Junit Testing environment, especially on the special cases, and the outcomes were positive everytime.

The classes that were put under test were IntMonomial, RealMonomial and Polynomial and all the important methods from each of them was tested separately.

# Conclusion

This project was first of all a great opportunity to remember and take a look again over the operations and processes that can be applied to polynomials and monomials. Even though implementing this kind of project has been assigned in the past programming courses too, I think now it required a more complex implementation, with all the functions.

More important than that, I learnt how to appply OO principles when modeling purely mathematical concepts. Although at first sight I was more tempted into going for an implementation based on array of coefficients, specific for structured programming, after thinking about the problem for a while I realized that an OO approach has many advantages and actually models the concepts of monomial and polynomial better, in individual and well-defined classes.

The importance of starting from a class diagram previously drawn also become very evident. As things become more complicated, a small inconsistency with the rest of the architecture can have a major impact and can lead to errors and problems that are not easy to solve.

# Further development

* Implementation for value at one point and root-finding
* Adding new possibilities for the input data format
* Add a button that copies the result of the last operation into one of the operands
* Improve the layout of the GUI
* Finding a more efficient implementation for multiplication and division

# Bibliography

<https://docs.oracle.com/javase/7/docs/api/overview-summary.html>

<https://www.tutorialspoint.com/java/>

<http://www.javatpoint.com/java-tutorial>

<http://stackoverflow.com/>