Polynomial Calculator

Autumn 2023

Max Champion

31. mars 2023

Innhold

1   Assignment objective

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

2 Problem Analysis, Modeling, Scenarios, Use Cases

2.1 Problem Analysis

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 integer power. A polynomial in a single indeterminate x can always be written (or rewritten) in the form :

$$\sum_{i=0}^{n}\left(a_i x^i\right) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n \tag{1}$$

where $a_0, a_1, a_2, ...$ an are constants and x is the indeterminate. The word "indeterminate"means that x represents no particular value, although any value may be substituted for it. The polynomial consists of a list of certain terms, also called monomials, for example $3x^2$ is a monomial.

The coefficient is 3, the indeterminate is x and the degree is 2. Forming a sum of several terms produces a polynomial, like the following one: $3x^2 - 5x + 4$, which has three terms with different exponents: the first is degree two, the second is degree one, and the third is degree zero. An alternative representation for polynomials consists of a sequence of ordered pairs:

$$(a_0 0), (a_1 1), (a_2 2), ..., (a_n n) \tag{2}$$

Each ordered pair $(a_i i)$ corresponds to the term $a_i x^i$ of the polynomial. An ordered pair is composed of the coefficient of the i-th term and its index representing the exponent i. For example, the polynomial $3x^2 + 2x + 1$ can be represented by the sequence $(3, 2), (2, 1), (1, 0)$. The sequence contains the (coefficient, exponent) pair of each monomial from the given polynomial.

So, we can say that polynomial is composed of one or more monomials $a_i x^i$ with i in the range $[0, n]$ where n is the degree of the polynomial. This way of representing polynomials can be used to perform the most common operations on polynomials: addition, subtraction, multiplication, division, differentiation, and integration.

2.2 Modeling

The user will be able to use the functions of the calculator by introducing in the interface, two polynomials. He/she will have to fetch the two polynomials so the application can store the values in "ordinary polynomials" then he/she will can choose a specific operation to perform, such as:
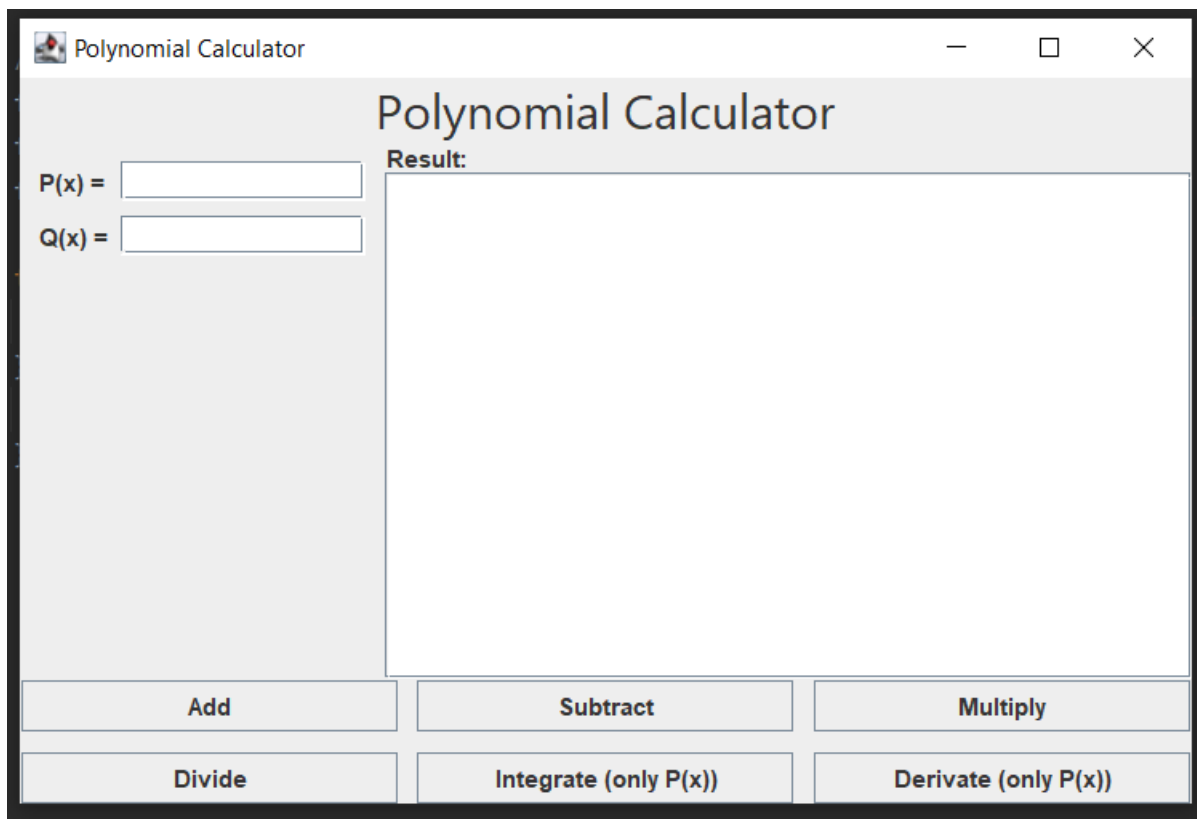
- Addition of two polynomials

- Subtraction of two polynomials

- Multiplication of two polynomials

- Division of two polynomials

- Differentiation of a polynomial

- Integration of a polynomial

The result of the chosen operation will be displayed in the interface. The user will also have the option to set the current result as one of the operands for the following operation, but he/she must fetch again the new operand, if the previous result is not a proper polynomial, the application will throw an error box.

2.3 Scenarios and use cases

A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal. The use cases are strongly connected with the user steps.

This is the reason why I tried to design my interface in a very friendly mode and the result is the following:


The user will introduce the two polynomials in the corresponding TextFields, and then he/she will fetch them so that the application will generate internally those two polynomials. If she/he wants to perform a certain operation, the user will have just to press the right button, and the result will be written in the result TextField. For some operations such as integration, division, derivation, the calculation is done only on the polynomial $P(x)$

3 Implementation

3.1 Diagrams

Uses cases diagrams :
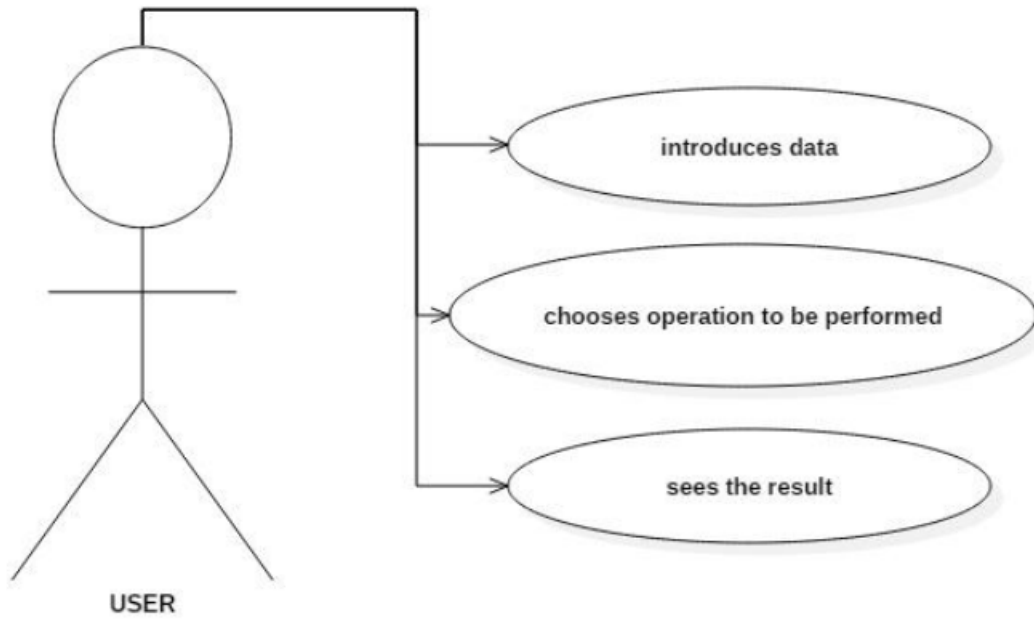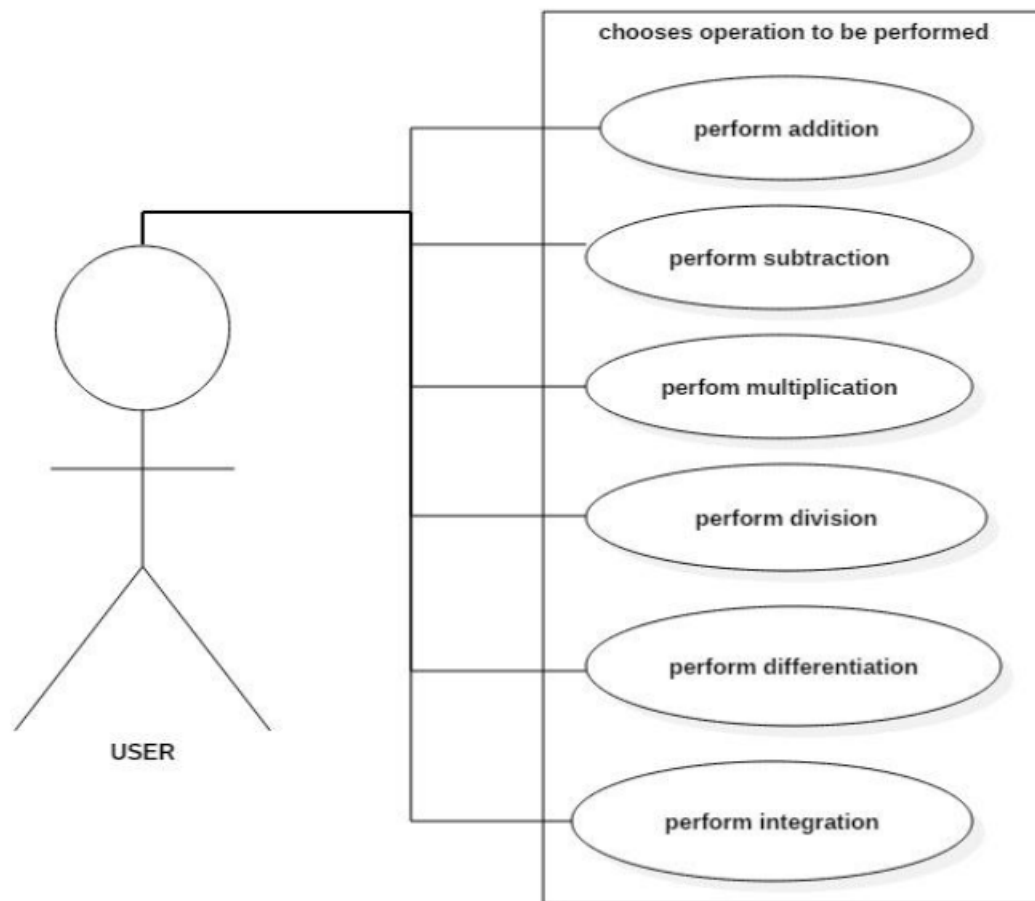
The use case presents the actor, which in our case is the user that interacts with the application. She/ He can perform several actions on the two chosen polynomials, such as addition, subtraction, multiplication, division, integration and differentiation.

ApplicationController

- view : SwingUI {readOnly}

+ ApplicationController(view : SwingUI)
+ run() : void
+ createPolynomial(textFieldString : String) : Polynomial
+ add(p : Polynomial, q : Polynomial) : Polynomial
+ subtract(p : Polynomial, q : Polynomial) : Polynomial
+ multiply(p : Polynomial, q : Polynomial) : Polynomial
+ divide(p : Polynomial) : Polynomial[]
+ derivate(p : Polynomial) : Polynomial
+ integrate(p : Polynomial) : Polynomial
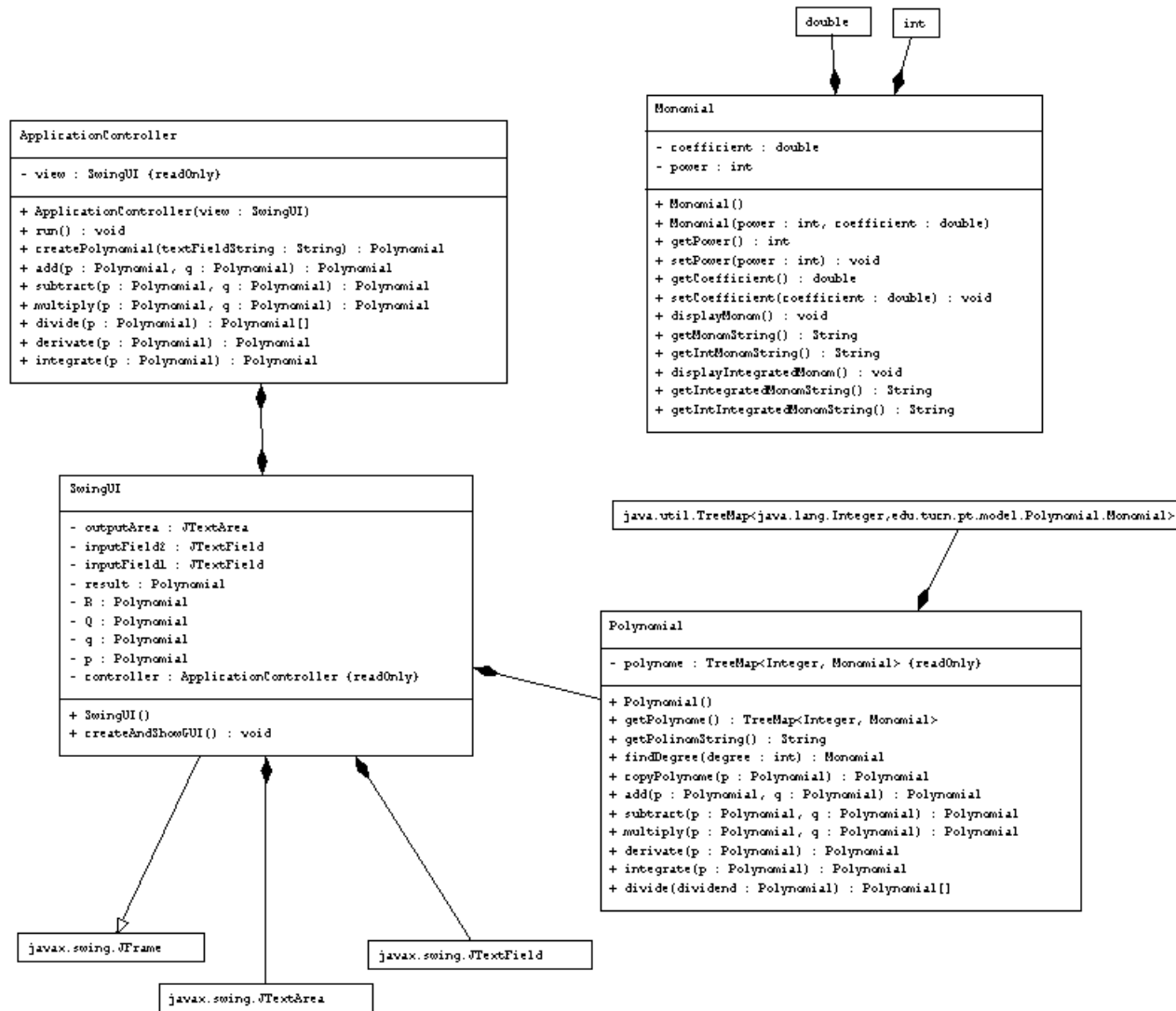
SwingUI

- outputArea : JTextArea
- inputField2 : JTextField
- inputField1 : JTextField
- result : Polynomial
- R : Polynomial
- Q : Polynomial
- q : Polynomial
- p : Polynomial
- controller : ApplicationController {readOnly}

+ SwingUI()
+ createAndShowGUI() : void

javax.swing.JFrame

javax.swing.JTextArea

javax.swing.JTextField

double   int

Monomial

- coefficient : double
- power : int

+ Monomial()
+ Monomial(power : int, coefficient : double)
+ getPower() : int
+ setPower(power : int) : void
+ getCoefficient() : double
+ setCoefficient(coefficient : double) : void
+ displayMonom() : void
+ getMonomString() : String
+ getIntMonomString() : String
+ displayIntegratedMonom() : void
+ getIntegratedMonomString() : String
+ getIntIntegratedMonomString() : String

java.util.TreeMap<java.lang.Integer,edu.turn.pt.model.Polynomial.Monomial>

Polynomial

- polyname : TreeMap<Integer, Monomial> {readOnly}

+ Polynomial()
+ getPolyname() : TreeMap<Integer, Monomial>
+ getPolinomString() : String
+ findDegree(degree : int) : Monomial
+ copyPolyname(p : Polynomial) : Polynomial
+ add(p : Polynomial, q : Polynomial) : Polynomial
+ subtract(p : Polynomial, q : Polynomial) : Polynomial
+ multiply(p : Polynomial, q : Polynomial) : Polynomial
+ derivate(p : Polynomial) : Polynomial
+ integrate(p : Polynomial) : Polynomial
+ divide(dividend : Polynomial) : Polynomial[]

3.2 Data structure

The data structures with which I have been working in this problem are either primitive data types, especially integers and doubles, and a more complex one, such as ArrayList type object or new created object such as Monom and Polinom.

Regarding this, I have decided to use ArrayList instead of the classic arrays because I think that they are more efficient from the point of view of memory management, performance and provide a faster access to their content, also, the size of an ArrayList is not fixed and adding new elements to the list is very easy because we do not have to worry about exceeding the "length" of the list (array).

A better implementation of the project would be to use Java Generics methods and classes, because they enable programmers to specify, with a single method declaration, a set of related methods or, with a single class declaration, a set of related types, respectively. In this specific case, creating a generic Monom class, would have been a better choice because the coefficients of a monomial can be integer or real numbers, so this would have made a cleaner code, also a Polinom generic class would have been needed too.

3.3   Packages

Java packages help in organizing multiple modules and group together related classes and interfaces.

In object-oriented programming development, model-view-controller (MVC) is the name of a methodology or design pattern for successfully and efficiently relating the user interface to underlying data models. The MVC pattern is widely used in program development with programming languages such as Java, Smalltalk, C, and C++.

The MVC pattern has been heralded by many developers as a useful pattern for the reuse of object code and a pattern that allows them to significantly reduce the time it takes to develop applications with user interfaces.

The model-view-controller pattern proposes three main components or objects to be used in software development:

- Model, which represents the underlying, logical structure of data in a software application and the high-level class associated with it. This object model does not contain any information about the user interface.

- View, which is a collection of classes representing the elements in the user interface (all of the things the user can see and respond to on the screen, such as buttons, display boxes, and so forth)

- Controller, which represents the classes connecting the model and the view, and is used to communicate between classes in the model and view.

My package :

- Application – contains a single class, which contains the customary main() method

- Model – contains the "brain" of the application, the classes which model the problem

  - Monomial
  - Polynomial

- View – contains a single class which represents the GUI.

- Control – it interconnects the model and the view

3.4   Class design

The whole idea of splitting your program into classes is based on a general rule named divide and conquer. This paradigm can be used almost everywhere: you divide a problem into smaller problems and then you solve these little, simple and well-known problems. Dividing your program into classes is one of the types of division which started to become common in last decade. In this programming paradigm we model our problem by some objects and try to solve the problem by sending messages between these objects. Because I followed the MVC architecture, my program consists of 4 parts:

3.4.1 Models

- Monomial class

  - A polynomial is composed by one or more terms, which in Mathematics are called monomials This class has two instance variables, an int exponent and a double coefficient

  - Constructor :
    * **public Monomial (int power, double coefficient)** : the constructor that initializes the monomials with the transmitted coefficient and exponent.
    * **public Monomial ()** : default constructor, initializes the degree/power and the coefficient with 0.

  - Methods :
    * **public int getPower ()** : return the power.
    * **public void setPower(int power)** : set the power.
    * **public double getCoefficient()** : return the coefficient.
    * **public void setCoefficient(double coefficient)** : set the coefficient.
    * **public void displayMonom()** : display the monom.
    * **public String getMonomString()** : returns one monomial as a string and it is used for the display in the GUI.
    * **public String getIntMonomString()** : return a monomial as a string, but with integer coefficients
    * **public void displayIntegratedMonom()** : displays the integrated monomial in the console.
    * **public String getIntegratedMonomString()** : returns the integrated monomial as a string.
    * **public String getIntIntegratedMonomString()** : returns the integrated monomial with integer coefficients as a string.

- Polynomial class

  - This class has only an instance variable which consists of a TreeMap¡Integer, Monomial¿ class that is used to keep the monomials of this polynomial

  - Constructor :
    * **public Polynomial()** : default constructor

  - Methods :
    * **public TreeMap¡Integer, Monomial¿ getPolynome()** : returns the actual polynomial, as an TreeMap of Monoms.
    * **public String getPolinomString()** : display the polynome in string.
    * **public TreeMap¡Integer, Monomial¿ getPolynome()** : return the polynome
    * **public Monomial findDegree(int degree)** : returns the monomial with the given degree.
    * **public Polynomial copyPolynome(Polynomial p)** : return a copy of the polynome.
    * **public Polynomial add(Polynomial p, Polynomial q)** : return the sum of the two polynomes.
    * **public Polynomial substract(Polynomial p, Polynomial q)** : returns the difference of the two polynome.

* **public Polynomial multiply(Polynomial p, Polynomial q)** : returns the product of the two polynome.
* **public Polynomial derivate(Polynomial p)** : return the derived of the polynome.
* **public Polynomial integrate(Polynomial p)** : return the integrated polynome.
* **public Polynomial[] divide(Polynomial p)** : return the devided polynome.
* **public Monomial getFirstMonom(Polynomial p)** : return the first monomial, the one with the greatest degree and non-zero coefficient, from a polynomial.

3.4.2 View

- SwingUI class

  – This class extends JFrame, I chose to do so because it is easier for me to set the characteristic properties like the DefaultCloseOperation, size, visibility, layout and to add graphic elements on it. I chose the GridLayout because it allows me to position my "elements" (labels, TextFields, buttons) however I want, by specifying the bounds of each element

  – Constructor :
     * **public SwingUI()** : the constructor that initializes the controller operarion.

  – Methods :
     * **public void createAndShowGUI()** : creates and shows the main window..
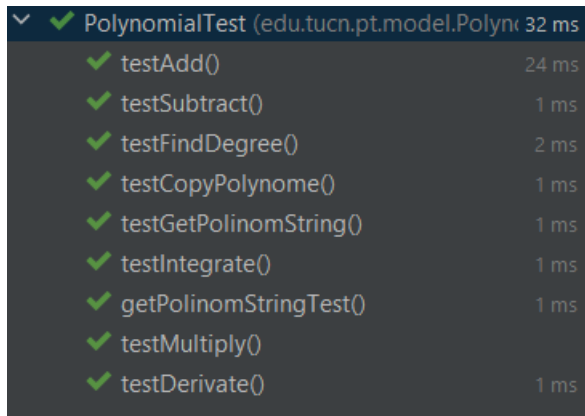
3.4.3 Controller

- SwingUI class

  – This is a very important class because it acts on both model and view. It controls the data flow into model object and updates the view whenever data changes

  – Constructor :
     * **public ApplicationController(SwingUI view)** : the constructor that initializes the controller operarion.

  – Methods :
     * **public void createAndShowGUI()** : creates and shows the main window..

3.4.4 Application

- App class

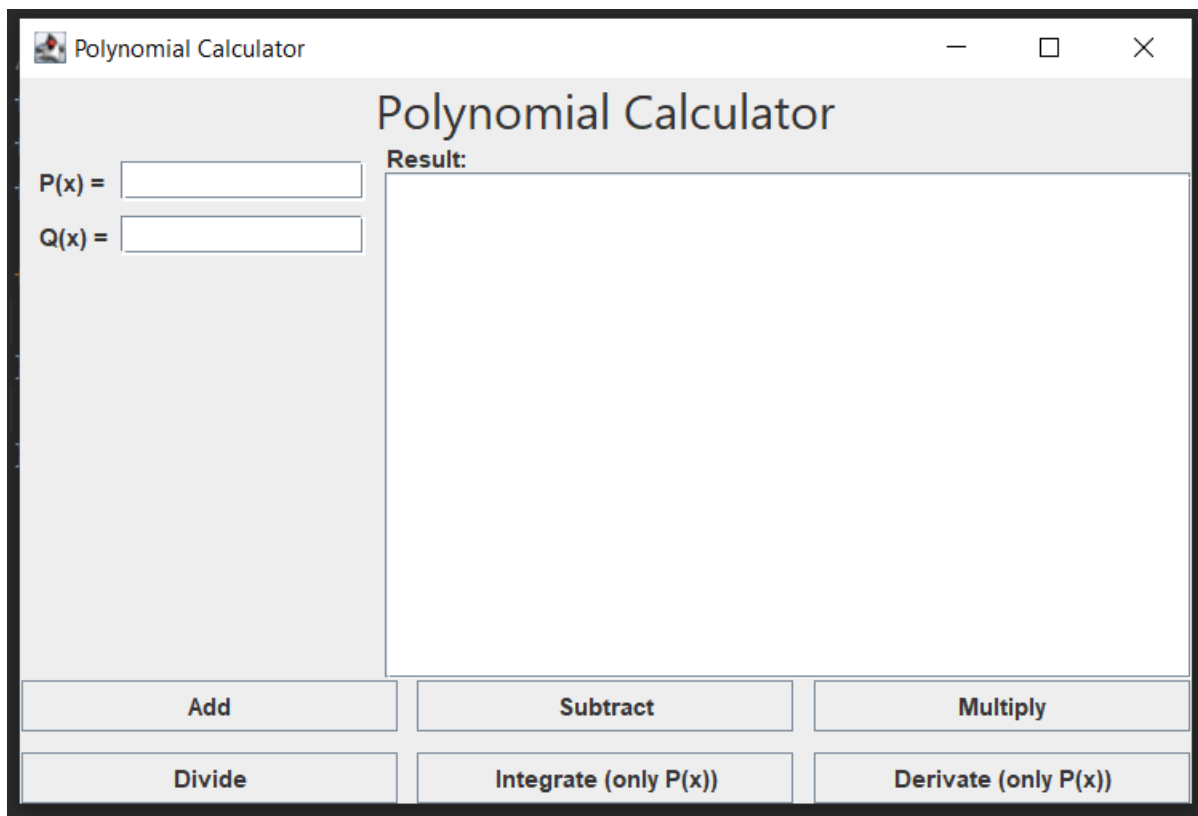 – This file contains the Application class which runs the main() method.

4 Testing

5 Result

This application was developed and tested only in Intellij, but this thing should not affect it's portability. From the point of view of the used algorithms, they are simple ones, taken from mathematics and implemented in Java

The application is an user friendly and helpful application to perform basic polynomial operations, as long as the user obeys the input conventions and it is familiar with polynomial operations such as: addition, subtraction, multiplication, division, differentiation and integration. As the application is developed on a Java platform, it is highly portable and allows it to run on several operating systems (as long as they have the Java SDK installed).

6 Conclusion

This project was a good exercise in remembering the OOP concepts learned in the first semester, but also learning new ones, I found it very useful and challenging at first. There are a few learned things which I would present next.

First of all , time management is very, very, very, crucial, because a good organizational spirit helps you see things gradually and making things from time helps you a lot.

Secondly, modelling the problem in a right way from the beginning helps you to implement it faster.

Thirdly, I arrived at the conclusion that facing problems with your code and trying to make it work by yourself, through the mean of research, has the benefit of learning new concepts and a better use of the known ones. In the end, one of the most important things that I have learnt is to make my interfaces from code.

7 Future improvements

Although at the moment I have chosen to implement just the basic polynomial operations, this application can be improved by adding new functionalities, such as:

- Compute the square of the polynomial (multiplication with self, separate button).
- Compute the value of a polynomial in a certain point.
- Find the roots of the polynomial (if the degree of the polynomial is ¡5).
- Plot the graphic of the polynomial.

8 Bibliography

- www.stackoverflow.com.
- www.wikipedia.org.
- www.whatis.techtarget.com/definition/model-view-controller-MVC.
- www.openclassrooms.com.