Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

System for polynomial processing

Ionuț Matiș

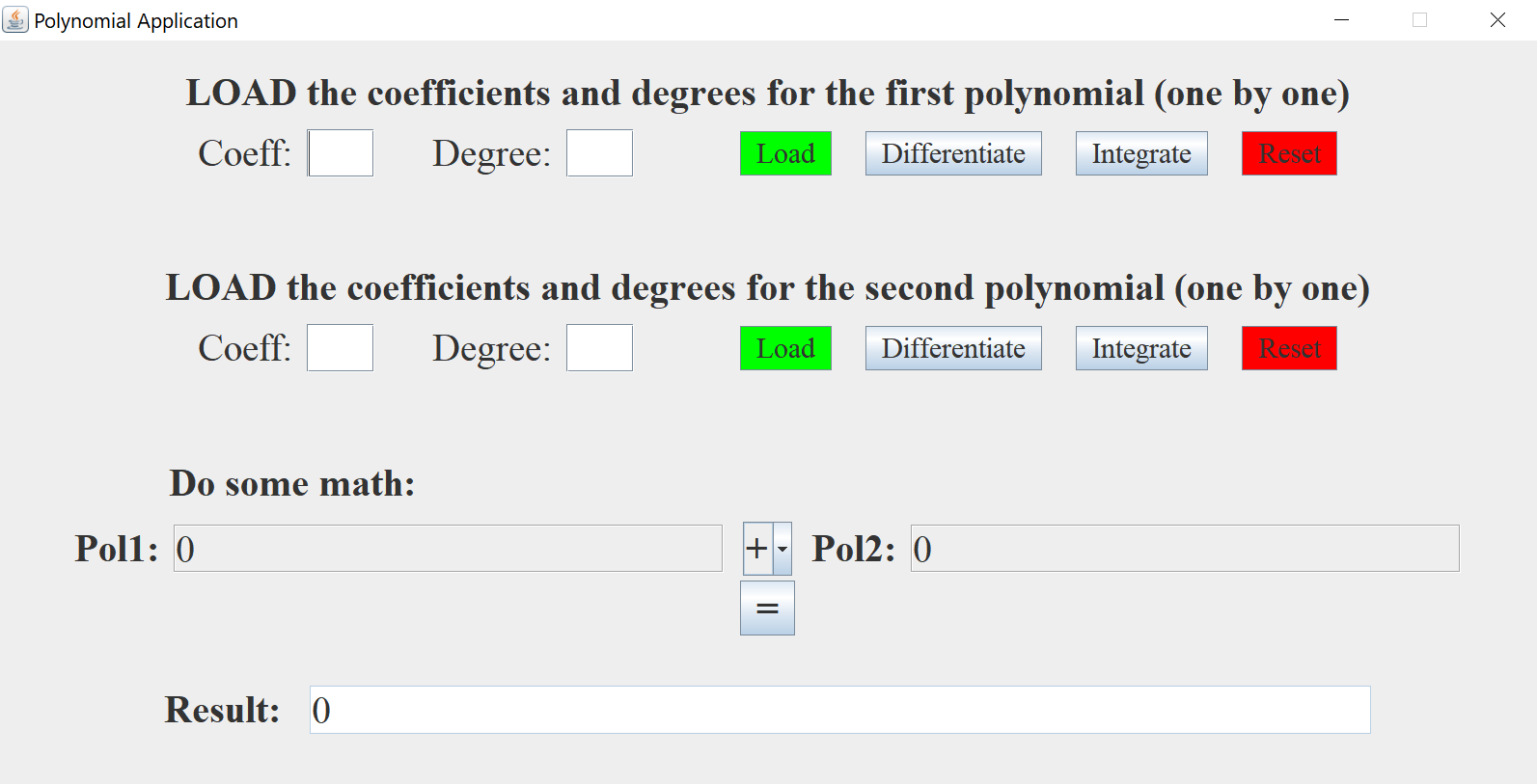
Group: 30421

# Task

* Main objective: design and implement a system which processes polynomials of one variable and integer coefficients;
* Secondary objectives:
  1. Implement the class for Monomial – see 4.1
  2. Implement the class for Polynomial – see 4.2
  3. Add the basic operations (addition, subtraction) in the Polynomial class – see 4.2
  4. Add the rest of the operations (differentiation, integration, multiplication, division) in the Polynomial class – see 4.2
  5. Create a GUI and implement a Model View Controller pattern – see 4.3
  6. Implement unit testing for this project using JUnit – see 5

# Problem analysis, modeling, use cases

The application is designed using IntelliJ IDEA and can perform all the required computations. It looks like this:



Usage:

a) **Insert the coefficient** (near the “Coeff” label) **and the degree** (near the “Degree” label) of the term of the polynomial.

b) After this, **press the green “Load” button.** You should see the term in the right of the “Pol1” and “Result” label (the term will be displayed as “a\*x^b”, where “a” is the coefficient and “b” is the degree). If you fail to insert a number (ex: **enter a letter or a non-number character**) and press “Load”, the result label will display an **“Invalid input”** message and the invalid term will not be added. If you want to add more terms, perform the previous operation with different a degree and coefficient (go to a).)

c) Now you should have a polynomial (ex: *a*n *x* n + *a*n−1*x*n−1 + ... + *a*2*x*2 + *a*1*x* + *a*0). You can **differentiate** it and get the result: n*a*n*x*n−1 + (n−1) *a*n−1*x*n−2 + ... + 2*a*2*x* + *a*1, you can **integrate** it and get the result: *a*n*x*n+1/(n+1) + *a*n−1*x*n/n + ... + *a*2*x*3/3 + *a*1*x*2/2 + *a*0*x* or you can **Reset** it and get a **zero polynomial** (if you want to insert a different polynomial.)

d) Insert the coefficients and the degrees of the second polynomial using the method described above (go to a) and insert the values in the second row.)

e) Now you should have **2 polynomials** (displayed in the right of “Pol1” label and “Pol2” label). Pressing on **“+”**, a drop-down menu will appear, and you can **select the operation** you want to perform on polynomials (**“+”** for addition, **“-”** for subtraction, **“\*”** for multiplication and **“/”** for division).

f) **Press on “=”** button for the result of the computation. This will be displayed in the right of the “Result” label. If you choose division, the result will be displayed in the following format:   
Q: <quotient as a polynomial> R: <rest as a polynomial>

Example

Load the following terms (coefficient and degree) for the **first polynomial: 3 4, -2 1, 5 0** and for the **second polynomial: 2 3, 1 1, 4 0**. The results will be:

* **Pol1:** +3\*x^4-2\*x+5
* **Pol2:** +2\*x^3+1\*x+4
* **Pol1 differentiation:** +12\*x^3-2
* **Pol2 differentiation:** +6\*x^2+1
* **Pol1 integration:** +3/5\*x^5-2/2\*x^2+5\*x
* **Pol2 integration:** +2/4\*x^4+1/2\*x^2+4\*x
* **Polynomial addition:** +3\*x^4+2\*x^3-1\*x+9
* **Polynomial subtraction:** +3\*x^4-2\*x^3-3\*x+1
* **Polynomial multiplication:** +6\*x^7+3\*x^5+8\*x^4+10\*x^3-2\*x^2-3\*x+20
* **Polynomial division:** Q: +3/2\*x R: -3/2\*x^2-16/2\*x+5

# Design

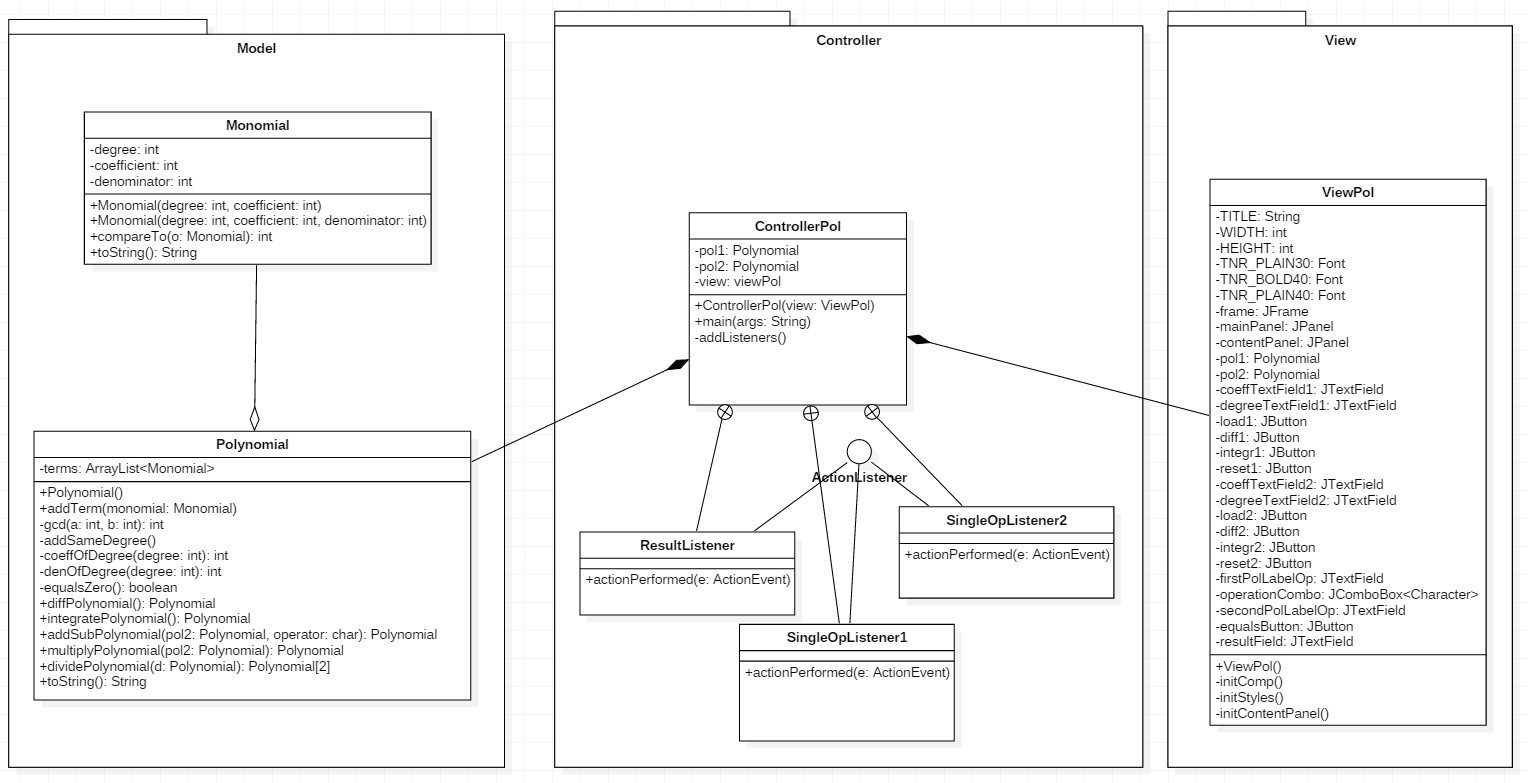
## 3.1 Decisions

Before creating such an application, it is very important to understand what a monomial is and how is relates to a polynomial. A monomial is structure that has a degree (integer, >= 0), a coefficient (integer) and a denominator (integer, != 0). We need the last attribute (denominator) when we perform the polynomial division or integration (ex: if we integrate 2\*x\*2 we will get as a result 2/3\*x\*3 – the first ‘3’ is the denominator.) In the other cases, when not explicitly assigned, the denominator is 1 and it is not written in the result.

I implemented this application so that a polynomial is an ArrayList of monomials, because it seemed to me to be the most logical approach and I could access each monomial separately when I want to perform various operations.

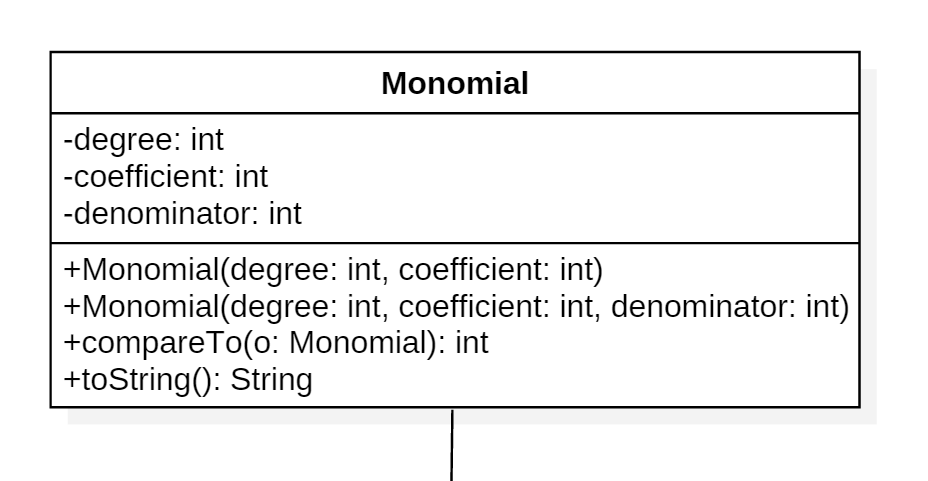
The **Model** contains the “knowledge” of the project – the classes Monomial and Polynomial with an aggregation relationship between them. The **View** contains the Graphical User Interface (what user sees when he/she opens the project), which consists of a Class that uses the Swing API and it represents the visualization of the data the model contains. The **Controller** acts on both model and view; it is very useful because it keeps the data and model separate. In this class I have 3 private inner classes that implement ActionListener. The first inner class (SingleOpListener1) is used for the buttons of the first polynomial, the second inner class (SingleOpListener2) is used for the buttons of the second polynomial and the third inner class (ResultListener) is used for the “=” button so that it performs the selected operation.

## 3.2 UML Diagram – the overview of the project



# Implementation

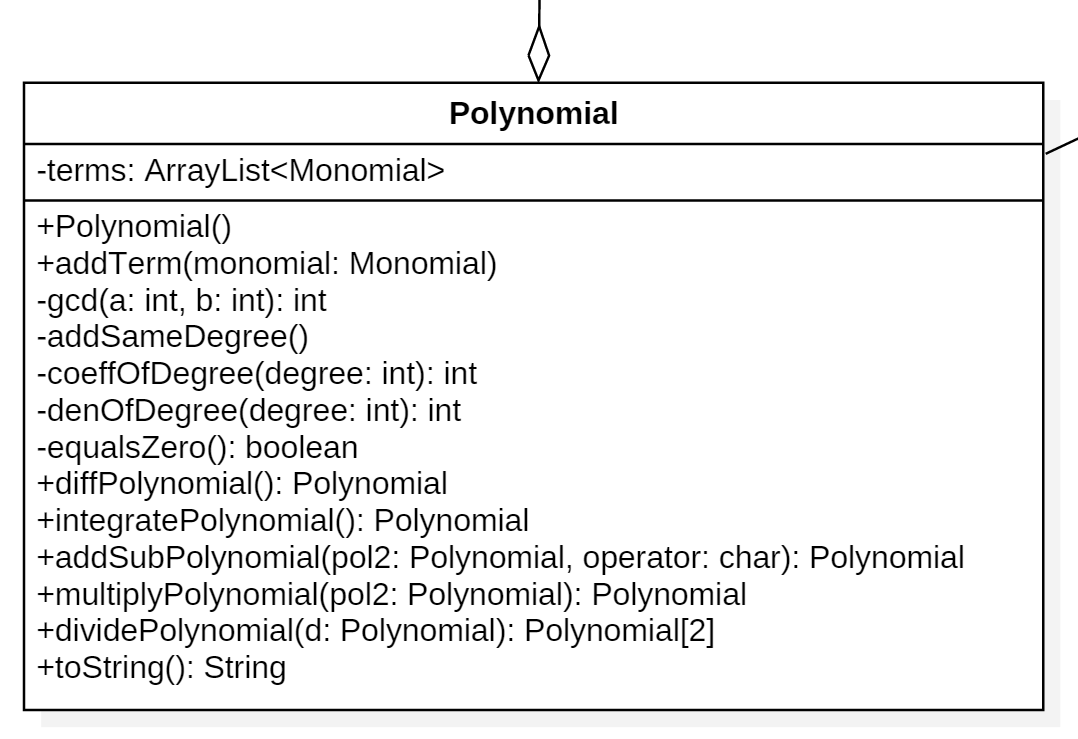
## 4.1 Monomial class



This class models the concept of a mathematical monomial. It has specific parameters (degree – degree of the monomial, coefficient - the coefficient of the monomial,  
denominator – denominator of the monomial).

* The **first constructor** creates an instance a monomial of a specific degree and coefficient. The denominator will be set to 1 as default;
* **compareTo(Monomial o)** - is used in the Polynomial class for sorting the monomials based on their degree (from the largest to the smallest).
* **toString()** - builds a string as a representation of a monomial so that the user can operate with it. There are special cases when the parameters are not displayed (ex: when the degree is 0 or denominator is 1.)

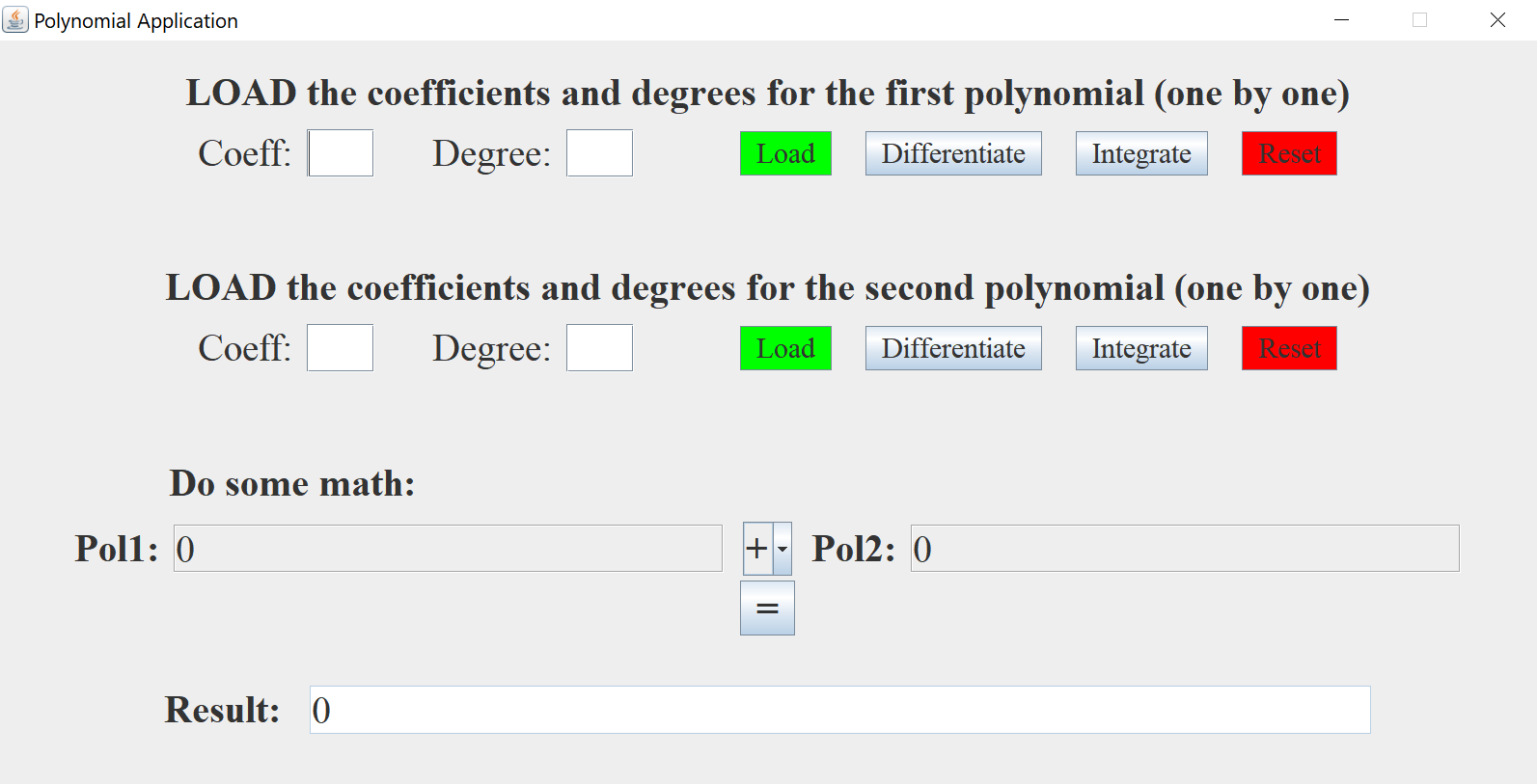
## 4.2 Polynomial class



This class models the concept of a mathematical polynomial. It contains an ArrayList of monomials and various methods specific to polynomials (ex: addition, division.)

* **addTerm(Monomial monomial)** - adds a monomial to the list of Monomials using the Collection method "add"; sorts the list so that the higher degrees are the first using the Collection method "sort"; adds the terms with the same degree using the method "addSameDegree()";
* **addSameDegree()** - adds the terms of the polynomial with the same degree and works even though the denominator is not 1. Before entering this method, the terms are sorted. The algorithm checks whether the consecutive terms have the same degree. If they do, the Smallest Common Multiple of the denominators is computed. After this, we apply the mathematical addition of the two fractions and remove the second consecutive element;
* **coeffOfDegree(int degree) / denOfDegree(int degree)** - returns the coefficient / denom of a specific degree;
* **diffPolynomial()** - used to differentiate the Polynomial. For each term of the polynomial it multiplies the coefficient with the degree and then decrements the degree;
* **integratePolynomial()** - This method is used to integrate the Polynomial. For each term of the polynomial it increments the degree, keeps the same coefficient and sets the denominator to the value of the degree.
* **addSubPolynomial(Polynomial pol2, char operator)** - This method adds or subtracts two polynomials based on the operator value (either '+' or '-'). It iterates through all degrees and checks whether the polynomials have coefficients of that specific degree. If yes, it adds or subtracts them using fraction addition or subtractions;
* **multiplyPolynomial(Polynomial pol2)** - multiplies every term of the current polynomial with very of the two polynomials by adding the degrees, multiplying the coefficients and multiplying the denominators. After this, adds the coefficients with the same degree;
* **dividePolynomial(Polynomial d)** - divides the current polynomial (this reference) to the polynomial d using the Horner mathematical method. See the method [here](https://en.wikipedia.org/wiki/Horner%27s_method%22);
* **toString()** - builds a string as a representation of a polynomial so, that the user can operate with it. It concatenates each monomial string representation in the ArrayList;

## 4.3 View class



This class represents the Graphical User Interface of the project. It is built on a single frame which holds a panel with BoxLayout.Y\_AXIS (aligns elements vertically). All the other elements are placed inside this panel on different panels with layout of type Flow.

On the first panel there is a label with a text. On the second there are 2 labels, 2 text fields and 4 buttons (for single operations on the first polynomial).

The next 2 panels are like the first 2 described above (with a label and several buttons, text fields for the second polynomial).

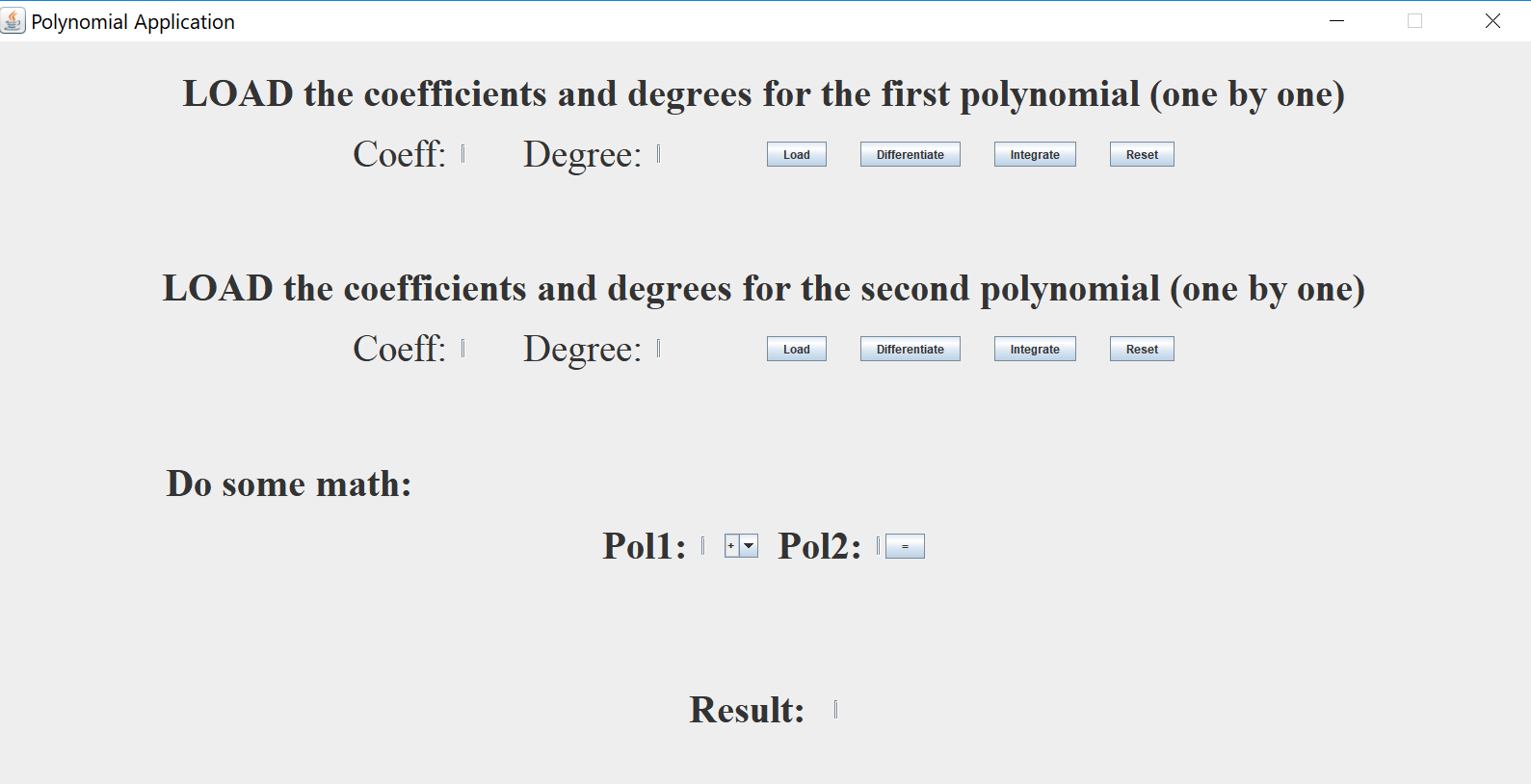
The fifth panel holds a label with a message "Do some math".

The sixth panel has 2 labels, 2 text fields, one JComboBox and one button (for operations performed on 2 polynomials)

The last panel displays the result and has one label and one non-editable text field.

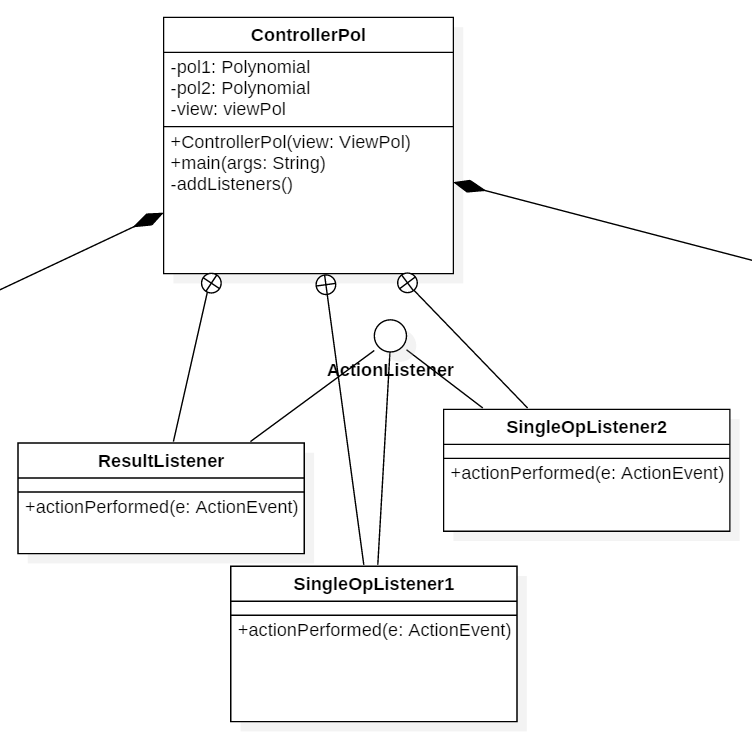
The GUI uses a font (Times New Roman) of different sizes and weights.

* The **constructor** initializes all elements (JButtons, Labels, JTextFields) and loads 3 methods **initContentPanel()**, **initStyles()** and **initComp()**;
* **initComp()** – sets the attributes for the frame (Title, Location, Width, Height etc);
* **initStyles()** - sets the fonts, sizes, borders and colors of the elements. Without this method, the project would look like:



* **initContentPanel()** -Adds, aligns the components on the various panels and then adds all the panels on the main panel. Lastly, adds this last panel on the frame.

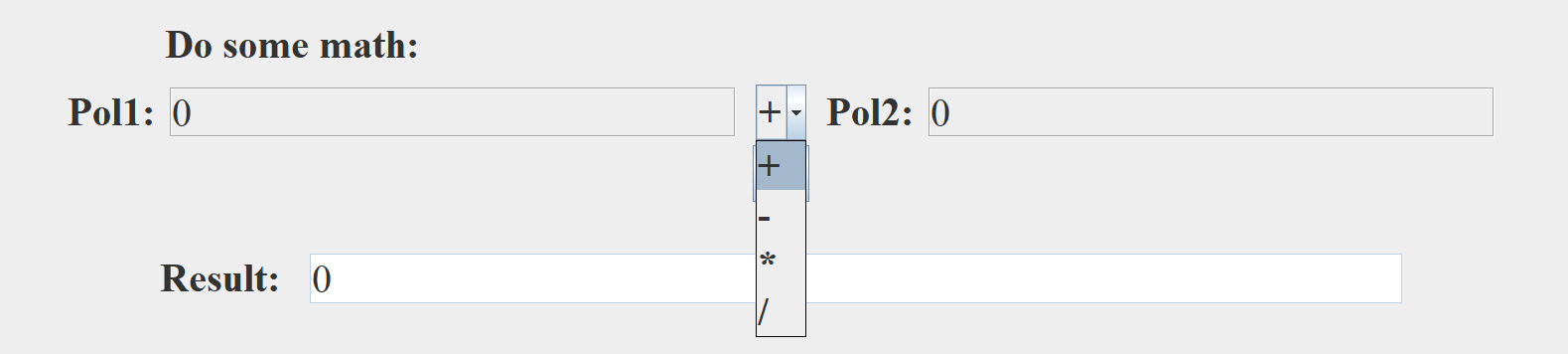
## 4.4 Controller class



This class represents the controller of the project. It interacts both with the view and model so that the classes of the project have well defined purpose. In this class we call methods from Model package and define listeners to the components of the View package.

* The **constructor** initialises the elements before any action is made (set the polynomials to be 0, sets the text fields accordingly to the initial status)
* **main(String args[])** – main method, the one from which the execution starts;
* **addListeners** – adds listeners (one of the 3 inner classes) to all buttons from the view;

I also have 3 private inner classes which implement ActionListener interface. The first inner class SingleOpListener1 controls the operations done on the first polynomial (Loading, Differentiation, Integration, Reset) by getting the actionCommand of the event. The second inner class SingleOpListener1 does the same as the the previous class, but for the second polynomial. The third class ResultListenes is an actionListener to the “=” button. It performs one of the following operations: addition, subtraction, multiplication, division based on the selected item (drop-down menu implementet as a JComboBox<Character>.



# Testing

We provide the input by loading in each term (coefficient degree) at a time, separated by commas (in our example). The first polynomial is +2\*x^3+1\*x+4, whereas the second one is +3\*x^4-2\*x+5. Differentiation and integration are done on the first polynomial.

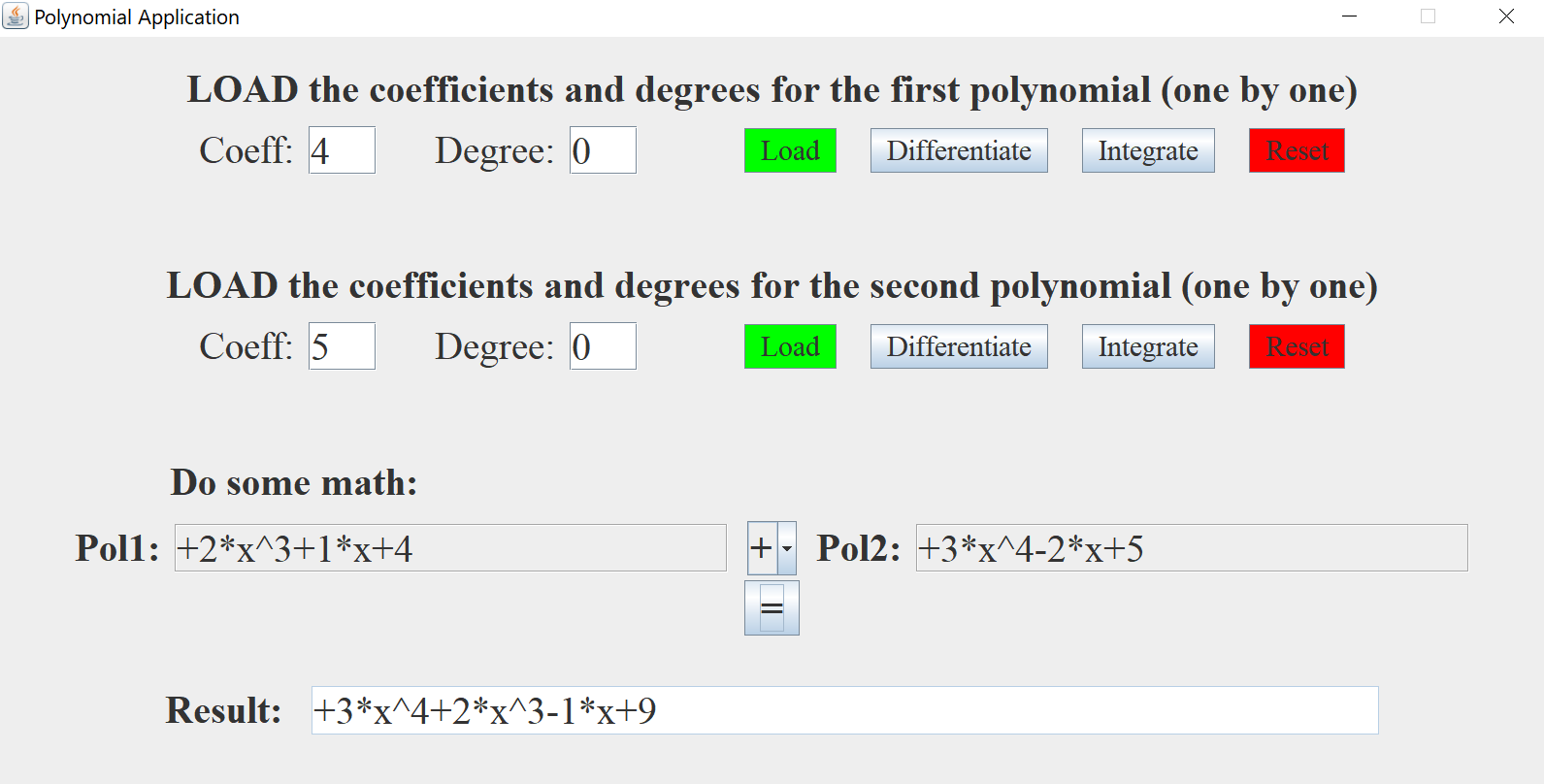
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method to be tested | Input | Expected result | Actual result | Pass/  Fail |
| addition | P1: 2 3, 1 1, 4 0  P2: 3 4, -2 1, 5 0 | +3\*x^4+2\*x^3-1\*x+9 | +3\*x^4+2\*x^3-1\*x+9 | PASS |
| subtraction | P1: 2 3, 1 1, 4 0  P2: 3 4, -2 1, 5 0 | -3\*x^4+2\*x^3+3\*x-1 | -3\*x^4+2\*x^3+3\*x-1 | PASS |
| multiplication | P1: 2 3, 1 1, 4 0  P2: 3 4, -2 1, 5 0 | +6\*x^7+3\*x^5+8\*x^4+10\*x^3-2\*x^2-3\*x+20 | +6\*x^7+3\*x^5+8\*x^4+10\*x^3-2\*x^2-3\*x+20 | PASS |
| division | P1: 2 3, 1 1, 4 0  P2: 3 4, -2 1, 5 0 | Q: 0  R: +2\*x^3+1\*x+4 | Q: 0  R:+2\*x^3+1\*x+4 | PASS |
| differentiation | P1: 2 3, 1 1, 4 0  P2: 3 4, -2 1, 5 0 | +6\*x^2+1 | +6\*x^2+1 | PASS |
| integration | P1: 2 3, 1 1, 4 0  P2: 3 4, -2 1, 5 0 | +2/4\*x^4+1/2\*x^2+4\*x | +2/4\*x^4+1/2\*x^2+4\*x | PASS |

# Results

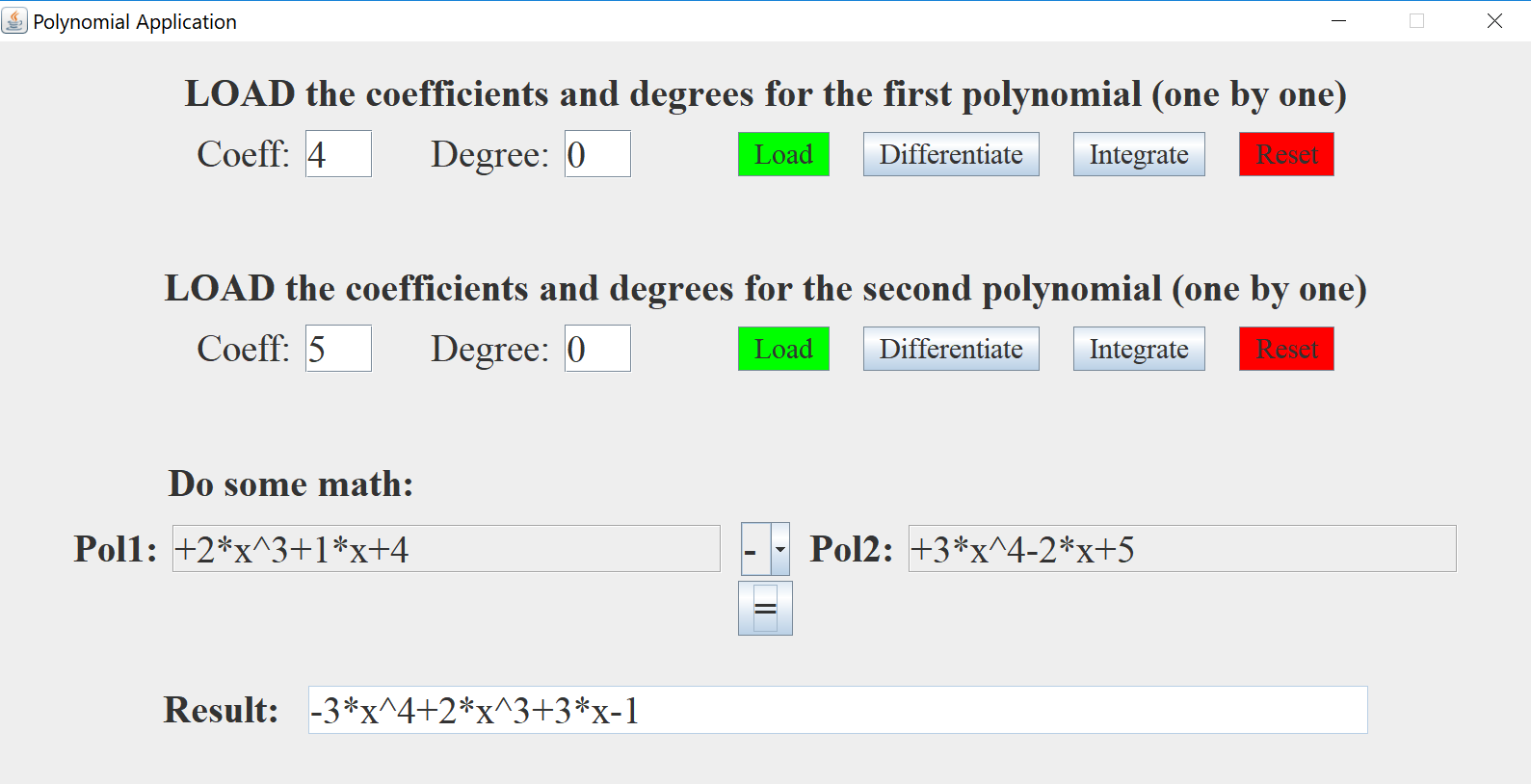
Consider the first polynomial: +2\*x^3+1\*x+4 (loaded term by term as above)

Consider the second polynomial: +3\*x^4-2\*x+5 (loaded term by term as above)

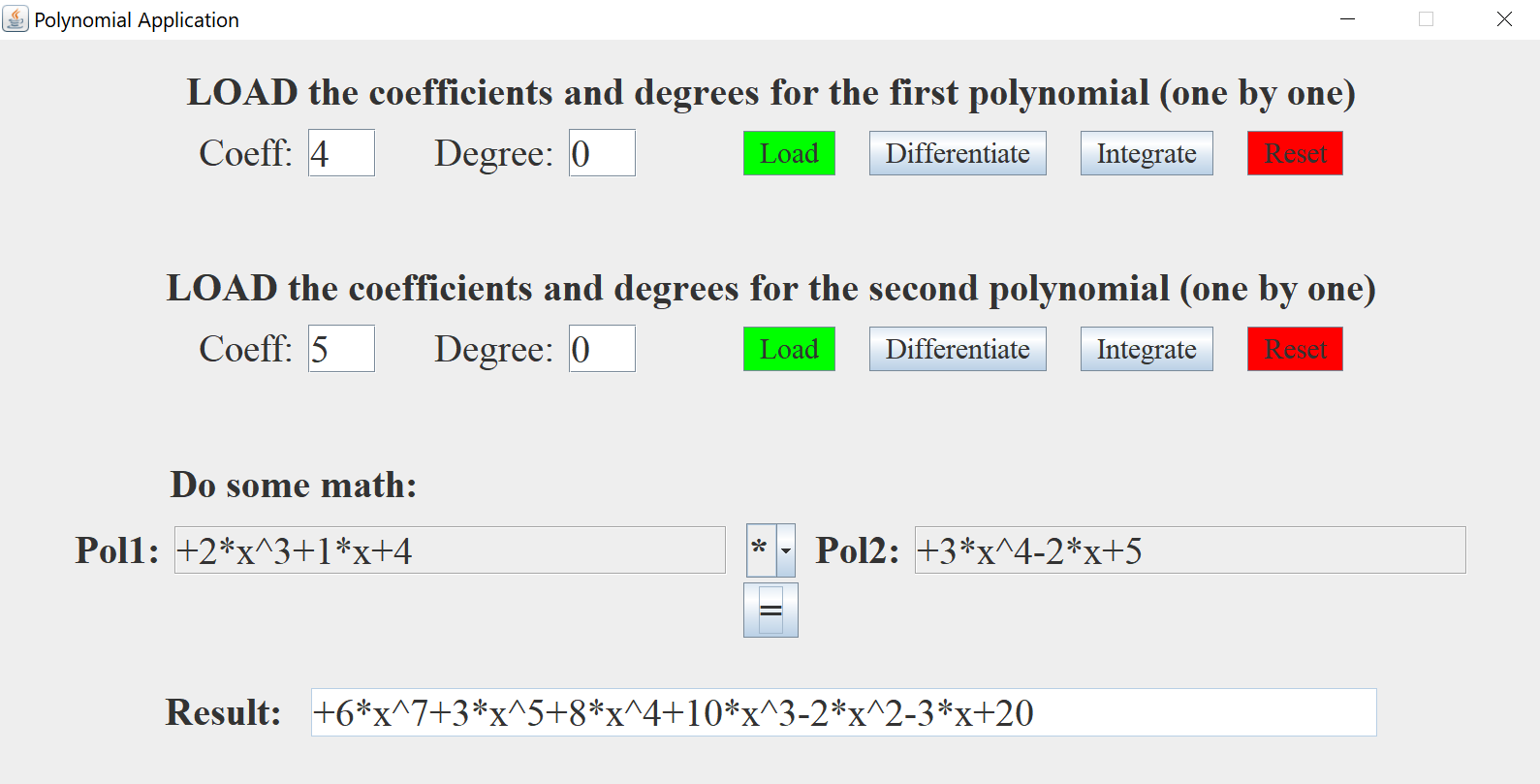
* Addition: the operation performs as expected;



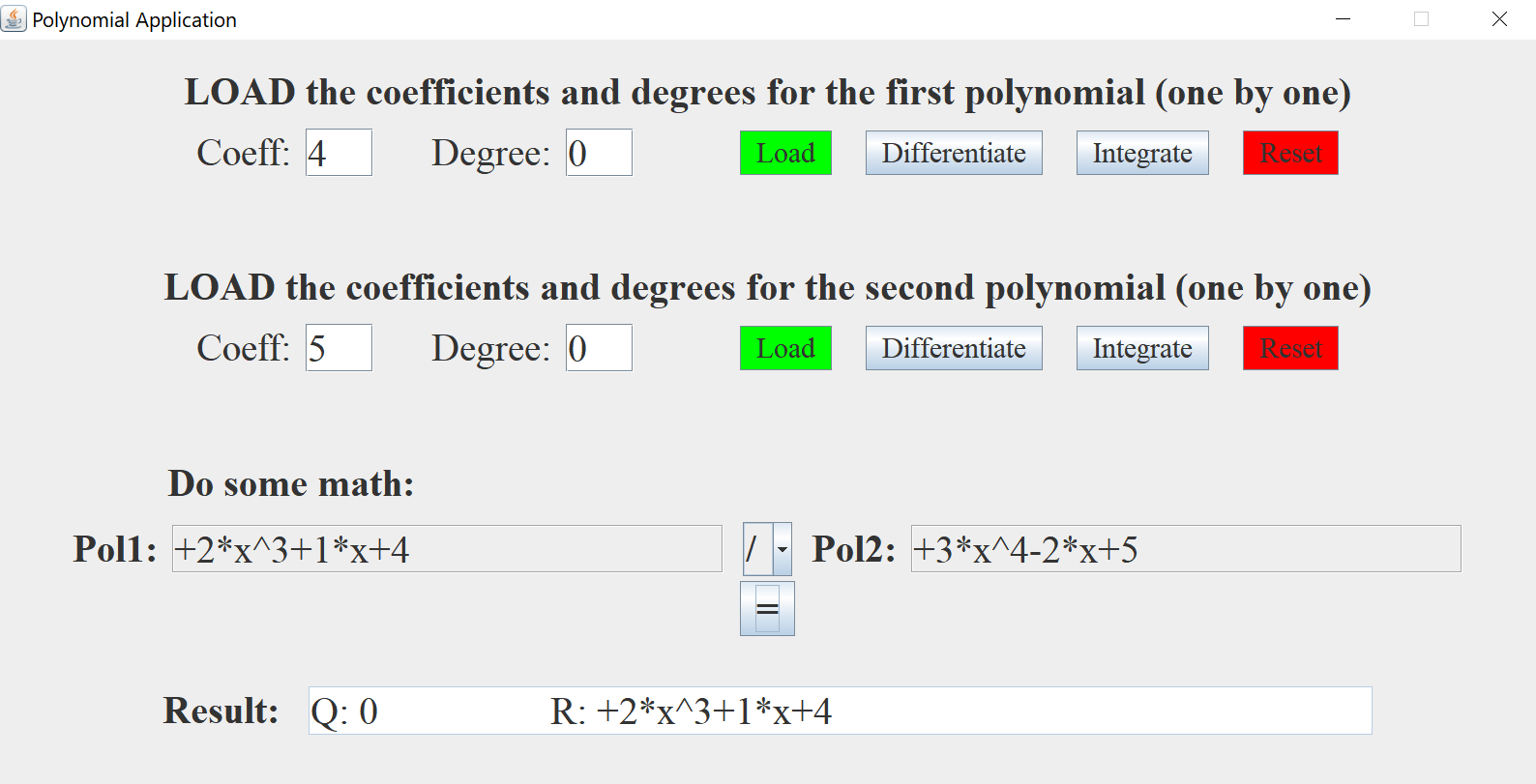
* Subtraction: the operation performs as expected;



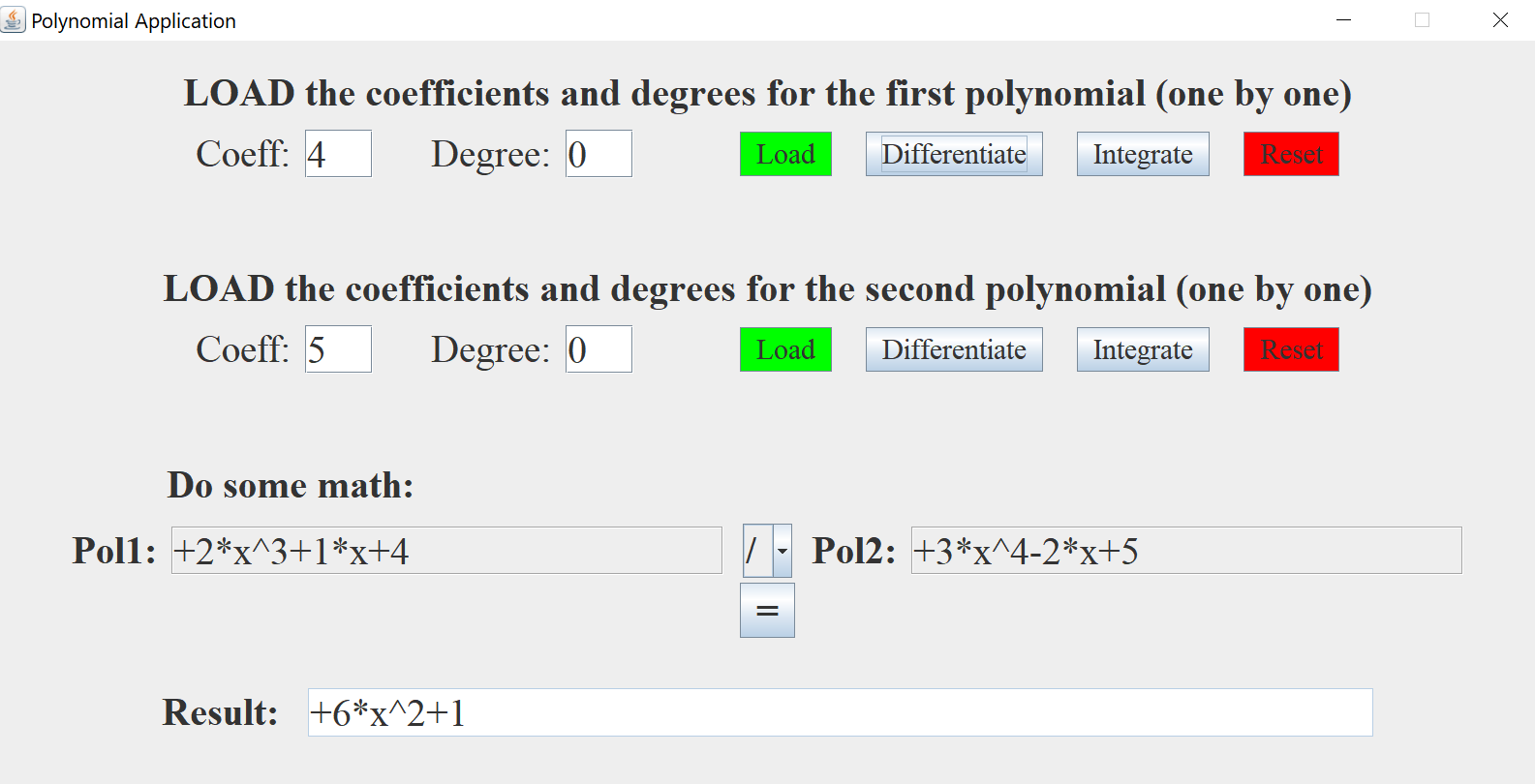
* Multiplication: the operation performs as expected;



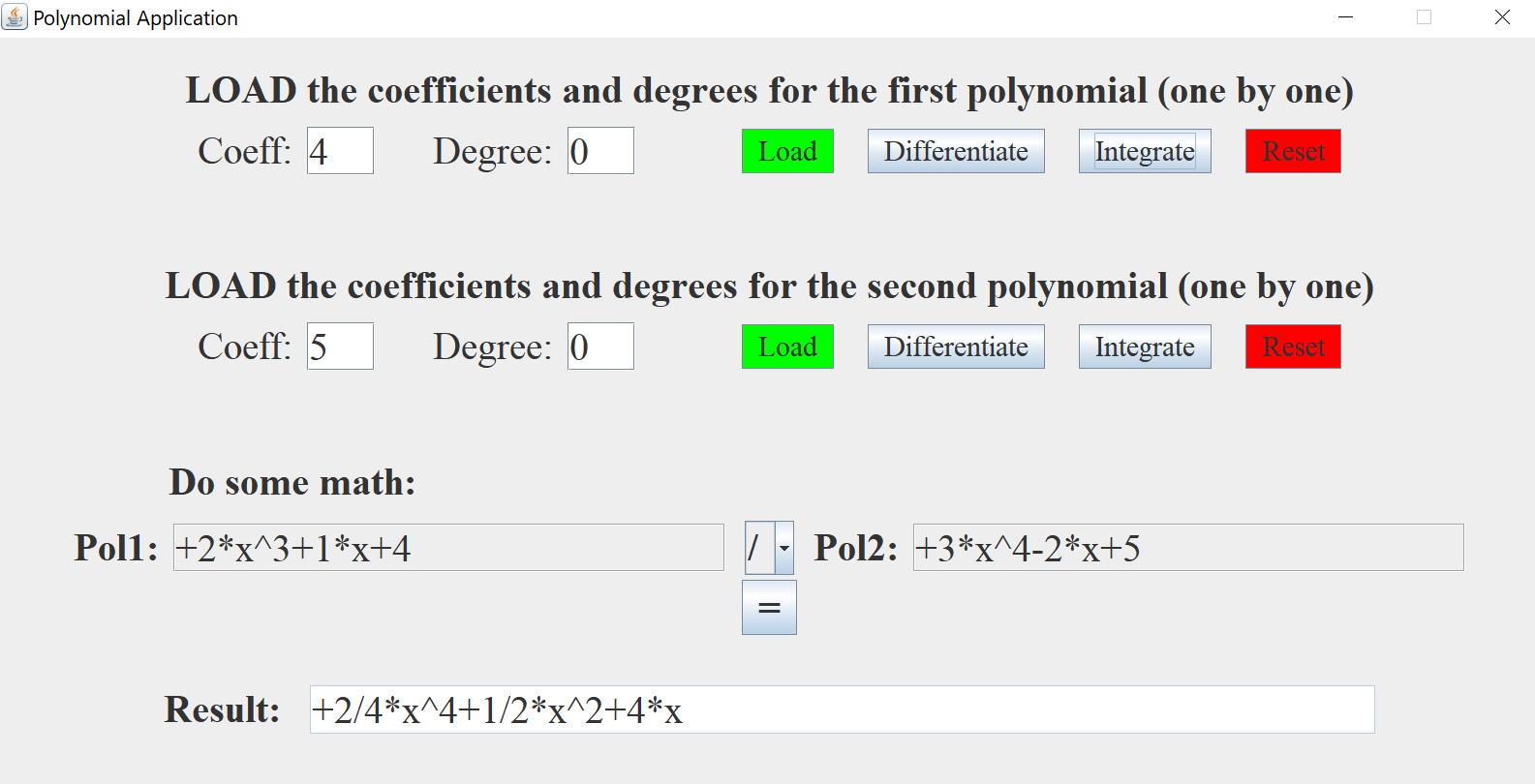
* Divison: the operation performs as expected;



* Differentiation: done on the first polynomial. The operation performes as expected;



* Integration: done on the first polynomial. The operation performes as expected;



# Conclusions

Implementing such an application requires some basic knowledge about polynomials and how to compute the operations between them, but also good handling of OOP paradigms.

I have learnt during this project how to create a Model-View-Controller pattern and I am sure it will be of great use on future applications and projects. I have also learnt how to test my classes and methods using JUnit and how to use Javadoc to generate API documentation in HTML format from Java source code.

Further improvements: read the polynomial directly from String format (using regex) and create a way of storing the entered polynomials, so that the user is not limited to only 2 such mathematical concepts.

# Bibliography

Wikipedia: <https://en.wikipedia.org/wiki/Polynomial_long_division>

JUnit: <https://www.tutorialspoint.com/junit/junit_basic_usage.htm>

Javadoc: <http://www.oracle.com/technetwork/java/javase/tech/index-137868.html>

StackOverflow: <https://stackoverflow.com/>