**FUNDAMENTAL PROGRAMMING TECHNIQUES**

**ASSIGNMENT 1**

**POLYNOMIAL CALCULATOR**

Anghel Dan-Marian

Group 30422

Professor: Diana Balc

**Table of contents:**

1. **Objectives**
2. **Analysis**
3. **Development**
4. **Implementation**
5. **Results**
6. **Conclusion**
7. **Bibliography**

**1. Objectives**

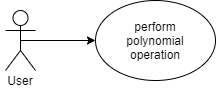
**Primary objectives:**

Design and implement a polynomial calculator with a dedicated graphical interface through which the user can enter polynomials, select the operation to be performed (i.e. addition, subtraction, multiplication, division, derivative, integration) and display the result. Each polynomial should be considered as a list of monomials of form “aX^p”, where “X” is the only variable, and the coefficients are positive integers.

**Secondary objectives:**

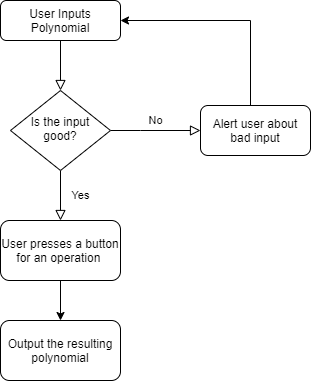
* **Use-cases** (Chapter 2):  a **use case** is a list of actions or event steps typically defining the interactions between a role and a system to achieve a goal, usually represented trough a flow-chart or an UML diagram.
* **Diagrams** (Chapter 3): they provide a mock-up for the project so that the developer can more easily grasp the concept and the small details of it.
* **Data structures** (Chapter 3): the types of data structures used in making the application.
* **The classes used** (Chapter 3): the project has been split in smaller classes that can accomplish certain tasks, accordingly to the MVC model.
* **Algorithms** (Chapter 3): certain algorithms have been used in order to make the polynomial operations possible.
* **GUI** (Chapter 3): an intuitive user interface is needed in order for the end-user to be able to manipulate the application easily.
* **Implementation** (Chapter 4): each class will be explained, along with the graphical interface.
* **Testing** (Chapter 5): some test classes are provided in order to test the functionality of the application. The Junit framework will be used to facilitate the resources for the testing.

**2. Analysis**



(\* Use-case diagram of the program)

|  |  |
| --- | --- |
| Use Case | Perform polynomial operation |
| Actor | End-User |
| Basic Flow | The user has access to a text box containing the rules to how to format the polynomials so that the program can function well. Next the user will enter the polynomials in the indicated text fields, followed by pressing the button corresponding to the desired operation. After this, in another text field will appear the result of the operation. |
| Alternate Flow | In the case of bad input, the |



(\* Flow chart of use-case description)

**3. Development**

The project follows the model of MVC and has the 3 classes (model, view, controller) accordingly. Aside from these we have classes for monomials and polynomials. According to the requirements, each monomial contains a power and a positive integer coefficient. The polynomials are based on an Array List of monomials. This has been the choice of data structure due to its simplicity, speed and versatility.

The long division algorithm has been used in order to implement the polynomial division, accordingly with the following pseudo-code:

**function** n / d **is**

require d ≠ 0

q ← 0

r ← n // At each step n = d × q + r

**while** r ≠ 0 **and** degree(r) ≥ degree(d) **do**

t ← lead(r) / lead(d) // Divide the leading terms

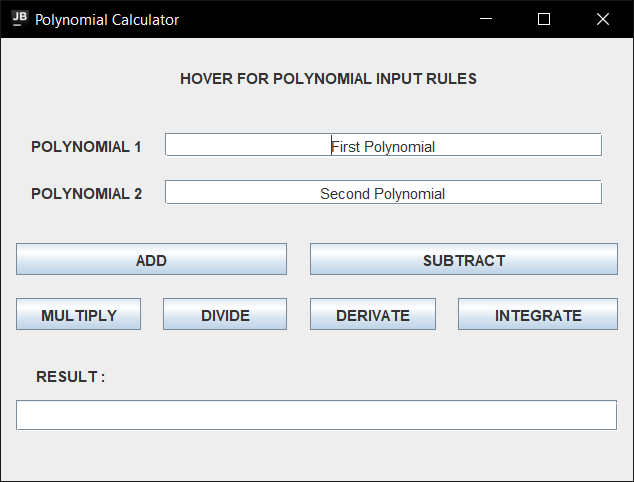
q ← q + t

r ← r − t × d

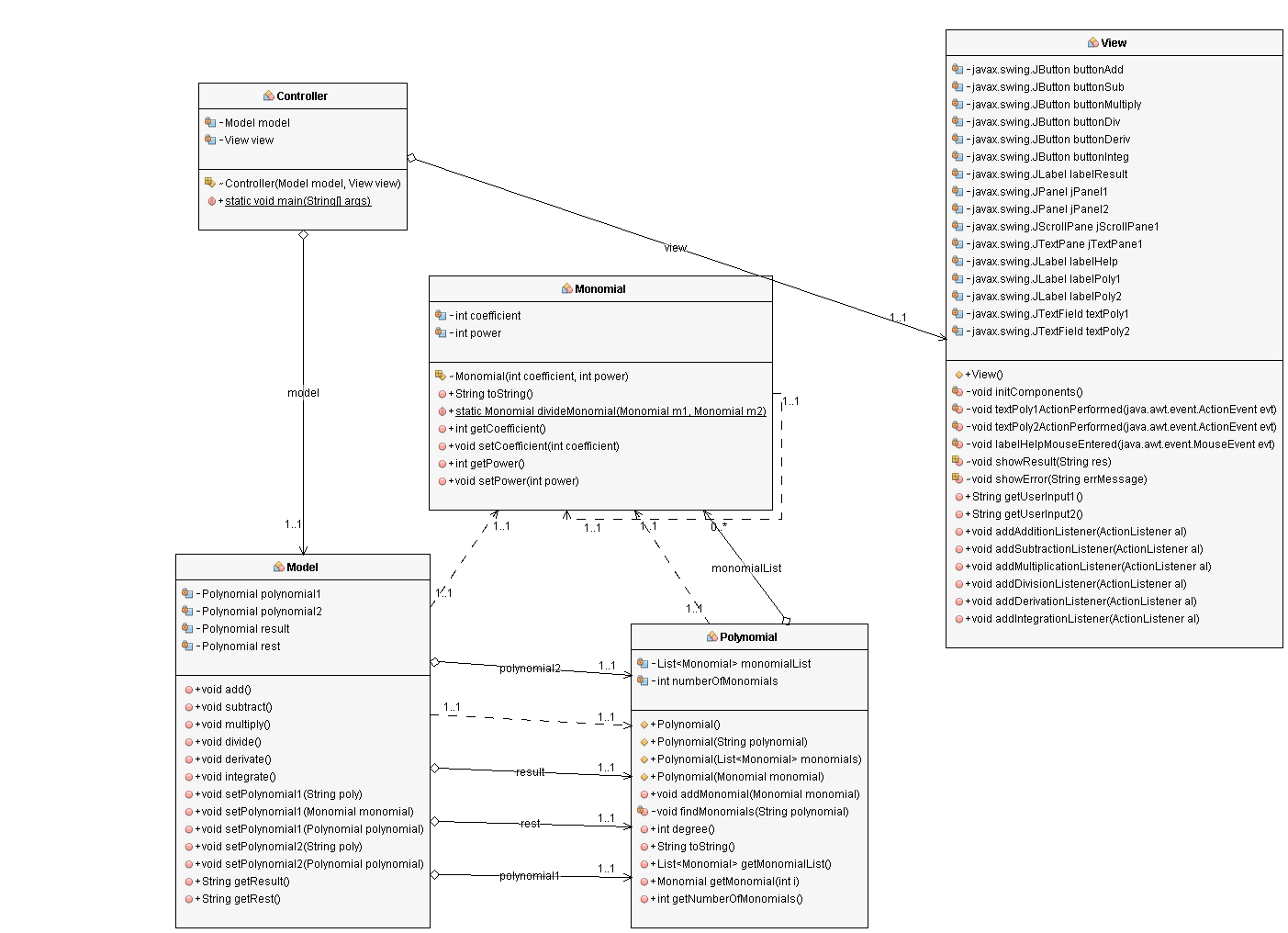
**return** (q, r)

(\* Long division algorithm, Source: Wikipedia)

The GUI is kept simplistic in order to provide a smooth experience for the user.



(\* image of the graphical user interface)

This is the class diagram of the project, where it can be observed the MVC, monomial, polynomial structure described above. 

(\* Class diagram of the project)

This image has been generated using the easyUML plugin of the NetBeans IDE.

A better-quality representation can be observed at the link: <https://imgur.com/a/W5aTuY9>

**4. Implementation**

(\* describing each class and their functionality)

**4.1 Monomial**

This class is the foundation of the project and has the following instance variables: coefficient and power, both being positive integers.

It has a constructor in order to set the coefficient and power, a static method to divide two monomials and it overrides the “toString ()” method in order to present the monomials in a nicer form. Also, it contains setters and getters for the instance variables.

**4.2 Polynomial**

This class is based on an Array List of monomial objects and it has a counter for the number of monomials inside the list as an integer. It has multiple constructors depending on the type of data available, the most common one having a String as parameter which is processed by the “findMonomials ()” method. The less common ones can take a single monomial as input or a list of monomials. There is also a no argument constructor which makes a call to the “super ()” method, for when an empty polynomial is needed.

We have the following methods implemented for the polynomial class:

* *addMonomial ()*, which takes a Monomial as input and adds it to the list, incrementing the counter.
* *findMonomials ()*, uses the following RegEx pattern ***"(-?\\b\\d+)[xX]\\^(-?\\d+\\b)***" in order to extract the power and coefficient of each monomial from the polynomial input given as a String. Next it creates the right monomials and adds them to the list.
* *degree ()*, returns the degree of the polynomial which represents the highest power of a monomial of the polynomial.
* *toString ()*, this method overrides the “toString ()” method of the class “Object” in order to give a nicer form for the polynomial.

And the rest are getter methods for instance variables and such.

**4.3 Model**

The central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application. The model is responsible for managing the data of the application. It receives user input from the controller. Having only the model by itself provides the functionality needed to perform the polynomial operations regardless whether we lack the other two components or not.

Inside this class we can find four Polynomial type objects, two for the operands, one for the result and one that is used to store the rest of the polynomial division. Next the 6 operations are implemented, addition, subtraction, multiplication, division, derivation and integration. Each one of them takes the 2 polynomials of the class and performs some arithmetics on them, followed by storing the result inside the corresponding polynomial. Exception to this rule is the derivation operation and integration, respectively. They only require one operand and will take as input from the user only the polynomial from the first text field. A particularity of the division operation is that it will print both the quotient and the rest of the operation in the “result” text pane like so: “Quotient + Rest”.

The rest of the methods inside the model are some setters for the two polynomial operands, tailored for different situations depending of the type of available input, and some getters to return the result and rest polynomials.

**4.4 View**

  The view means presentation of the model in a particular format. A view is a (visual) representation of its model. It would ordinarily highlight certain attributes of the model and suppress others. It is thus acting as a *presentation filter*. The view is effectively, the bridge between the user and the model, it represents the application as a whole to the user, but without the extra functionality of the controller it is effectively just a mock-up.

First of all, we have the swing elements that are presented to the user, making up the graphical interface. The first thing the user sees will be a label that upon hovering over, will create a pop-up that will instruct the user as to how to

format the input so that the application can do its job. Next, there are the two text fields for the user to feed the polynomial inputs to. These are marked by appropriate labels and placeholder text. Below these, there are the 6 buttons corresponding to each operation, marked by text. And lastly, the “result” text field that will give the user the output of the chosen operation.

Secondly, there are methods for initializing the swing components, show the result in the text pane, or show a message dialog with an error, get input from the user for the two polynomials, and lastly methods for adding listeners for the buttons corresponding with the operations.

**4.5 Controller**

  A controller is the link between a user and the system. It provides the user with input by arranging for relevant views to present themselves in appropriate places on the screen. It provides means for user output by presenting the user with menus or other means of giving commands and data. The controller receives such user output, translates it into the appropriate messages and pass these messages on to one or more of the views.

It takes a model and a view as instance variables and initializes them inside the constructor whilst also setting up the listeners. Inside the controllers there are sub-classes for each action listener for each polynomial operation. They listen for user input and set the result inside the model accordingly with the operation selected. Otherwise, it shows an error message if the input from the user is not valid. There is also the main () method to initialize the components and run the application.

**5. Results**

Junit 4.12 has been used in order to test the polynomial operations with success.

Firstly, a testing class has been created with methods for each operation. Each operation has 2 methods, one to test a result that we know is correct and one to test a wrong result.

Secondly, a runner class was implemented to test each method and to print to the console the result of the test, and in case of failure it will the expected and the actual result along with the exact discrepancies between the 2 values.

**6. Conclusions**

The polynomial calculator is a great example for the MVC model since it really shows the strengths of this developing method despite its simplicity. It has great potential for further development by implementing more operations and polynomials with more variables. Certain polynomial presets, such as *(x+1) ^2* or *x^2-1*, can be added via buttons or a drop-down menu to help the user input data more easily.

There could be added support for polynomials with non-integer coefficients and/or powers, such as fractions, square roots or even

complex numbers. For the latter we can implement methods that compute the real and imaginary part of a complex number

And for a more ambitious project, a plotting system could be added so that the user can visualize the polynomials on the axis

**7. Bibliography**

* Class diagram of the project: <https://imgur.com/a/W5aTuY9>
* EasyUML plugin for generating class diagrams inside NetBeans: <http://plugins.netbeans.org/plugin/55435/easyuml>
* Wikipedia article about polynomial long division algorithm: <https://en.wikipedia.org/wiki/Polynomial_long_division>
* Easy to use website to generate UML diagrams, flowcharts, etc.: <https://app.diagrams.net/>