DOCUMENTATION

Polynomial Calculator

ASSIGNMENT *1*

STUDENT NAME: Oltean Darius-Ionuț
GROUP: 30224

CONTENTS

1. ASSIGNMENT OBJECTIVE

The main objective of this project is to design, implement and test a polynomial calculator. To achieve this I had to go through the following steps:

- Problem analysis, modeling solutions and scenarios and build use cases will be described in detail in the second section.
- OOP design of the application along with the UML package and class diagrams, in section three.
- In the fourth section it will be described the implementation of each class and important methods contained therein.
- Section five will be dedicated to the result and test cases of the application.
- In the last section I will describe a briefly conclusion and possible future developments.

2. PROBLEM ANALYSIS, MODELING, SCENARIOS, USE CASES

A polynomial contains a coefficient, exponent and a variable. Because a variable is always present and depends on its own exponent, we can exclude it from the data structure. The exponent(key) and the coefficient(value) can be stored as a map of Integers and Doubles.

The input is validated with a regex and after the validation it's parsed with the help of another regex which divides the input in monomials. In the end each monomial it's inserted in the map.

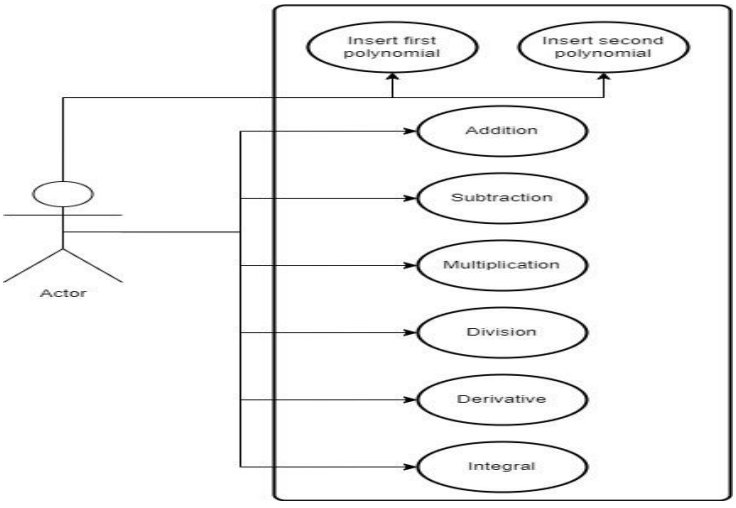
The operations can be implemented in a class and called by the main logic class in a switch case.

As a constraint the client has to introduce a text which contains a polynomial with a specific pattern, a coefficient only if it is different than one or zero, a 'x' variable if the exponent it is different than zero and '^' if the exponent is different than zero or one and an exponent. For example:

- x^2+4x^3-2x+7 – correct;
- 1x^3-x+7x^0 – wrong;

- **Use case description scenario:**
  - **Use case:** Operate on two polynomials.
  - **Primary actor:** Actor.
  - **Success scenario steps:**
    - Actor tries to introduce two polynomials.
    - Actor selects an operation.
    - The calculator checks if the introduced polynomials are valid.
    - The calculator performs the operations.
    - The calculator displays the operation.
  - **Alternative scenarios:**
    - **Invalid input:**
      - ✓ The calculator tells the actor that the input is invalid.
      - ✓ The scenario returns to the first step.

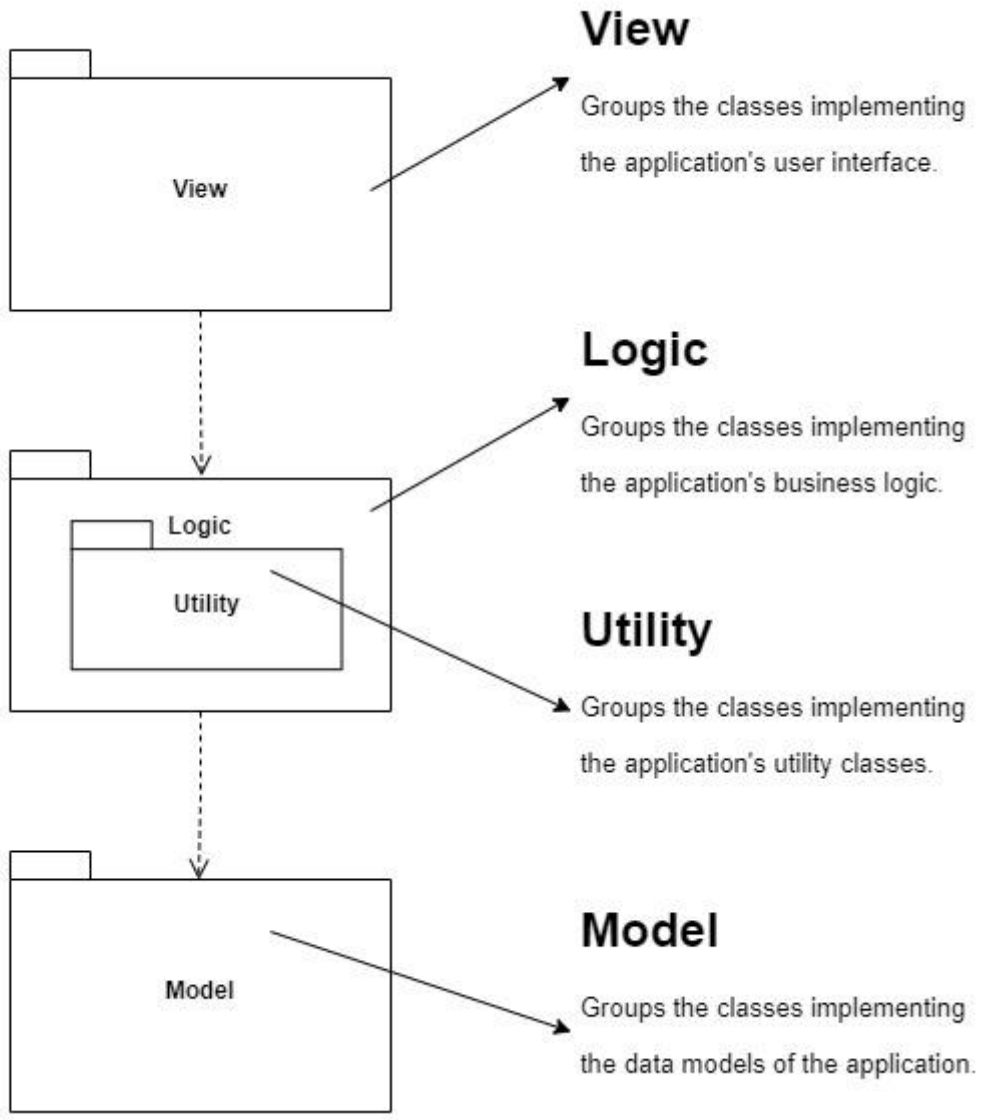
Use case diagram:



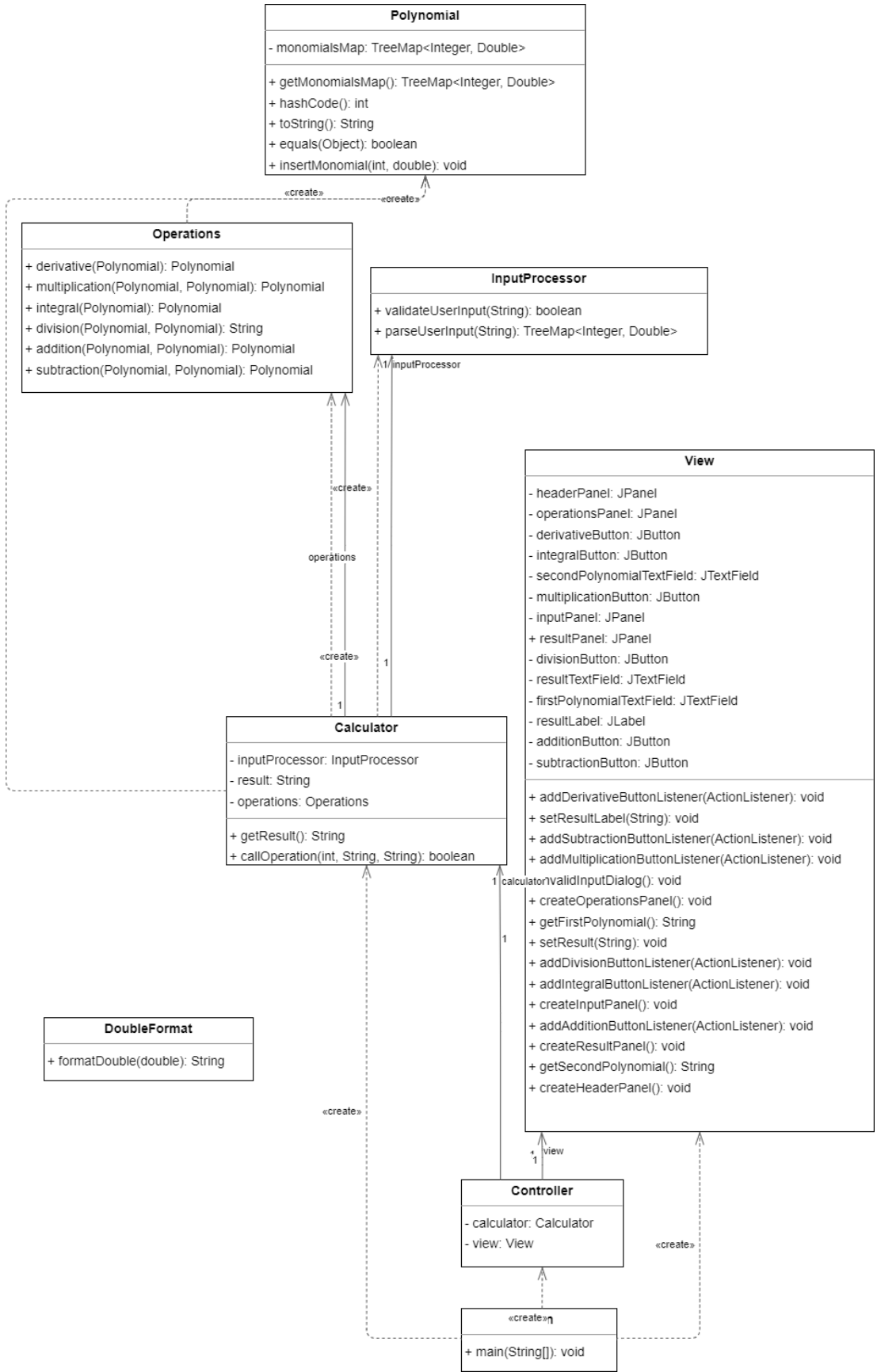
3. DESIGN

The OOP design of the application is structured in 4 packages, 3 main packages and one sub-package. The graphical user interface it is implemented in the View package which contains View class and Controller class. Data models are described in the Model package with the only existent class, Polynomial. The Utility sub-package, which has DoubleFormat class, is contained by Logic package, along with the Calculator class, Operations class and InputProcessor class, all these define the business logic of the application.

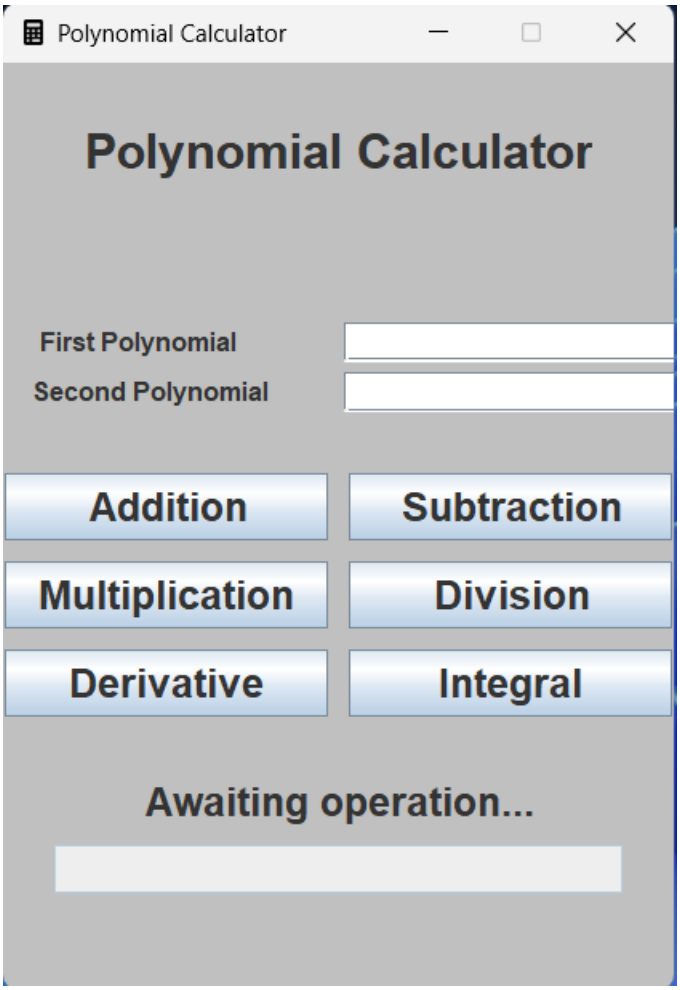
UML package diagram:



UML class diagram:



Graphical user interface:



4. IMPLEMENTATION

- *Polynomial class*

  Represents the data model of the application which contains a single field, a TreeMap of Integers and Doubles.

  It has two constructors, one that receives an already created TreeMap and one which initialize the TreeMap field, and a method to insert monomials into the TreeMap. It also overrides the equals and hashCode methods, for testing purposes, and toString method.

- *Operations class*

  Has no fields but only methods for each operation: addition, subtraction, multiplication, division, derivative and integral.

Each method receives two already processed Polynomial objects, except for derivative and integral methods which receive only one object.

All methods, except division, returns a Polynomial object containing the result, however the division method also returns the result but as a String.

The algorithms are based on crossing both or just one objects where applicable and performing the necessary operations.

```
public Polynomial derivative(Polynomial polynomial) {
    Polynomial returnValue = new Polynomial();

    for (Integer exponent : polynomial.getMonomialsMap().descendingKeySet()) {
        if (exponent != 0)
            returnValue.insertMonomial( exponent: exponent - 1,  coefficient: polynomial.getMonomialsMap().get(exponent) * exponent);
        else {
            returnValue.insertMonomial( exponent: 0, returnValue.getMonomialsMap().getOrDefault( key: 0,  defaultValue: 0d));
        }
    }

    return returnValue;
}
```

- *Calculator class*

Is the top-level element of the business logic part which also communicates with the graphical user interface and is in charge of passing input to the input processor for validating and parsing.

It has three fields, an Operations object, InputProcessor object and a result String.

In the constructor it initializes the operations and inputProcessor objects and also has a method, which receives a selector number for a certain operation and two input Strings, to call a operation and sets the result field.

- *InputProcessor class*

The purpose of this class is to validate and parse the user input through the help of regexes.

Has two methods, first one validates the input using following regex: (([+-])?([0-9]+)?(x)?(\^[0-9]+)?)+, second one parse the user input and returns a TreeMap containing the monomials.

```
public class InputProcessor {
    20 usages   Darius
    public boolean validateUserInput(String userInput) {
        if(userInput.isBlank())
            return false;
        userInput = userInput.trim();
        Pattern pattern = Pattern.compile( regex: "(([+-])?([0-9]+)?(x)?(\\^[0-9]+)?)+");
        Matcher matcher = pattern.matcher(userInput);

        return matcher.matches();
    }
```

- *DoubleFormat class*

  Is an utility class which helps formatting the coefficient in toString() method of the Polynomial class. Briefly it returns a string with the integer value of the coefficient if it is a integer or the double value with two decimals.

  Source of the implementation: https://stackoverflow.com/questions/703396/how-to-nicely-format-floating-numbers-to-string-without-unnecessary-decimal-0s

- *Controller class*

  Is the connection point between the business logic and graphical user interface of the application. It has two fields, a View object and a Calculator object.

  It implements six inner classes for each button listener, the implementation of classes receives the user input and use the calculator object to execute a certain operation then sets the result in the interface.

- *View class*

  Implements the graphical user interface and is divided into four panels, an header panel, an input panel, an operations panel and a result panel. It uses six buttons, one for each operation. For the input and input it uses text fields.

5. RESULTS

Besides the manual testing I introduced three unit tests using JUnit, one for operations, one for validating user input and one for parsing user input.

There are twelve tests for operations, six true tests for each operation and six false tests for each operation.

All of above tests passed.

6. CONCLUSIONS

- Conclusion:
  - The application is functional and can perform any required operation on polynomials.
- Learned skills:
  - I delved into JUnit and learned a new type of software architecture. I upskilled my abilities to organize my code.
- Future developments:
  - Possible features to implement: Replace the text from the buttons with a corresponding icon, performing operations on multiple polynomials, allowing user to introduce different variables not only ‘x’.

7. BIBLIOGRAPHY

1. *Regex:* Tutorials List - Javatpoint
2. RegExr: Learn, Build, & Test RegEx
3. Stack Overflow - Where Developers Learn, Share, & Build Careers
4. diagrams.net