

DOCUMENTATION

ASSIGNMENT *1*

STUDENT NAME: RUNCAN PAUL-MARIUS

GROUP: e_30422

CONTENTS

1.	Assignment Objective	3
2.	Problem Analysis, Modeling, Scenarios, Use Cases.....	4
3.	Design	7
4.	Implementation	13
5.	Results.....	18
6.	Conclusions.....	22
7.	Bibliography	22

1. Assignment Objective

Main Objective:

- *Design and implement a polynomial calculator with a graphical user interface which allows the user to insert a polynomial using the keyboard (the accepted format of each monomial can be : freeTerm, x , $x^{(exp)}$, $(coef)x$, $(coef)x^{(exp)}$), select the operation and view the result (Reference 1.1)*

Secondary Objectives:

- *Analyze the problem and identify the requirements*
- *Design the polynomial calculator*
- *Implement the polynomial calculator*
- *Test the polynomial calculator*

2. Problem Analysis, Modeling, Scenarios, Use Cases

Functional requirements:

The polynomial calculator should:

- *allow the user to insert a polynomial from the keyboard*
- *allow the user to select an operation(add, subtract, multiply, divide, integrate, derive);*
- *print an error message if the polynomial has a wrong syntax*
- *perform the operation chosen;*
- *print the result from the operation;*

Non-Functional requirements:

The polynomial calculator should:

- *be intuitive, it must be clear for the user where the polynomials should be inserted and how to choose and compute the operation.*
- *be reliable, failures should appear only if the bounds of Integer number aren't respected.*
- *be maintainable, issues should be easily fixed because the code is well-structured and is based on simple algorithms.*

Use Cases:

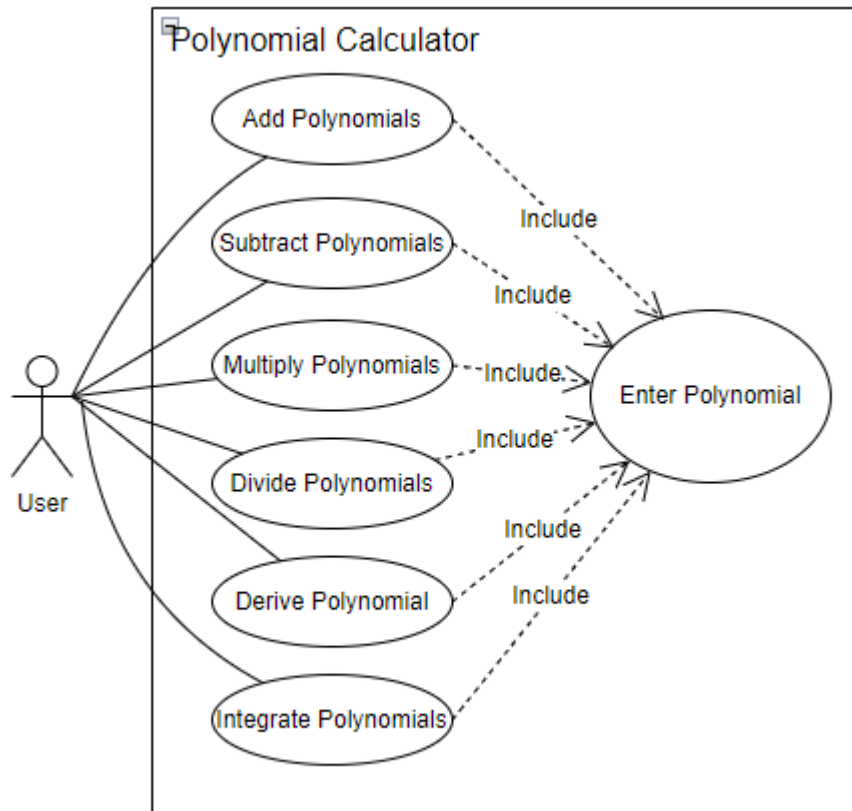


Figure 1. Use Case diagram

Use Case: Addition, Subtraction, Multiply

Primary actor: user

Main Success Scenario:

1. The user inserts 2 correct format polynomials in the text fields associated with them in the graphical user interface. The format is specified in ([Reference 1.1](#)).
2. The user selects one of the three operations.
3. User presses the “Compute” button
4. The calculation is performed and the result is displayed on the Label associated with the result.

Alternative Sequence:

1. The user inserts a wrong format polynomial. (Different sequence than in ([Reference 1.1](#))).
2. The application displays an error message.
3. The scenario return to step 1.

Use case: Division

Primary actor: User

Main Success Scenario:

1. The user inserts 2 correct format polynomials in the text fields associated with them in the graphical user interface. The format is specified in ([Reference 1.1](#)).
2. The user selects the “Divide” operation.
3. The user presses the “Compute” button.
4. The calculation is performed and the result is displayed (Quotient on the Result Label, Remainder on the Remainder Label).

Alternative Sequence:

1. The user inserts a wrong format polynomial.(Different sequence than in ([Reference 1.1](#))).
2. The application displays an error message.
3. The scenario return to step 1.

Use case: Derivation, Integration.

Primary actor: User

Main Success Scenario:

1. The user inserts a correct format polynomial in the text field corresponding to the first polynomial. The format is specified in ([Reference 1.1](#)).
2. The user select one of the 2 operations.
3. The user presses the “Compute” button.
4. The calculation is performed and the result is displayed on the Result Label.

Alternative Sequence:

1. The user inserts a wrong format polynomial.(Different sequence than in ([Reference 1.1](#))).
2. The application displays an error message.
3. The scenario return to step 1.

3. Design

Level 1: Overall System Design

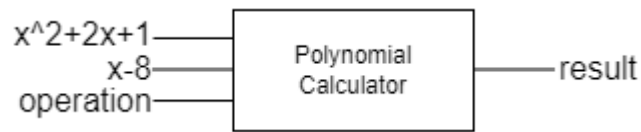


Figure 2. System Black Box

The system design consists of 3 inputs (First Polynomial, Second Polynomial and Operation input) and one output in case of addition, subtraction, multiplication, derivation and integration, and 2 outputs for division.

Level 2: Design in sub-systems/packages

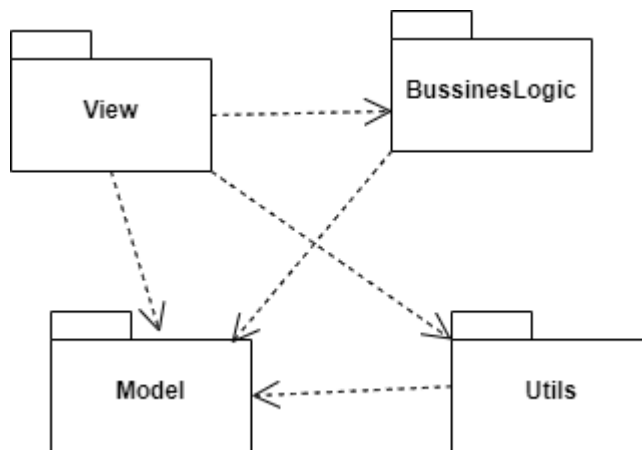


Figure 3. Package Diagram

View: contains the classes that implement the Graphical User Interface and Validator Class/Interface for the input.

BusinessLogic: contains the Operations class in which only static methods are declared and operations are performed using this class.

Model: contains the Polynomial class where the modelling of the polynomial as a Map is realized.

Utils: contains method considered as helpers, such as, a class that contains a method which turns the string read from the GUI into a Map and a class that helps in keeping a number to the Integer format if it is an Integer and to the float format if it is a float.

Level 3: Division into classes/Class Diagram.

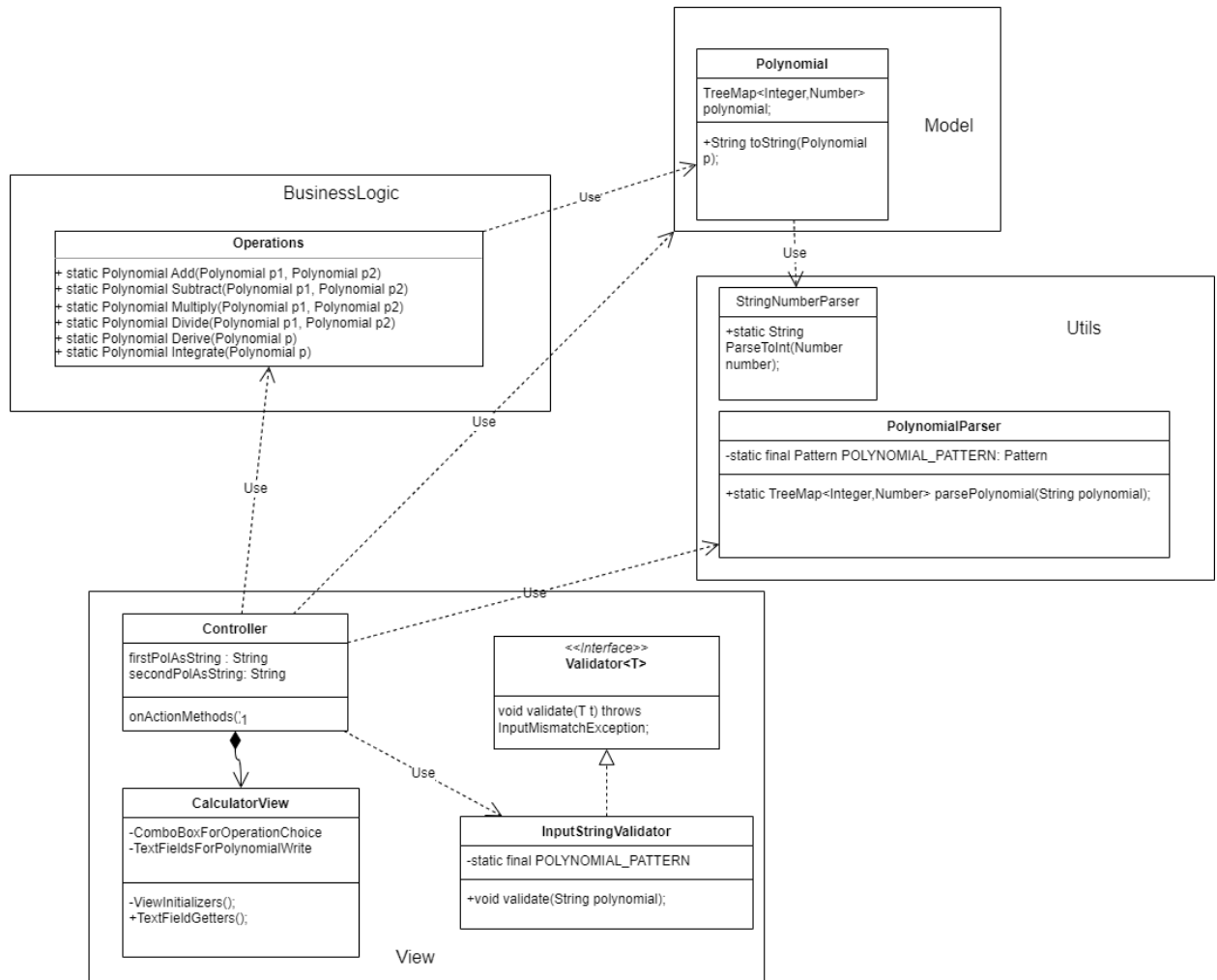


Figure 4. Class Diagram

Used Data Structures:

The main data structure used is Map, in depth, a `TreeMap`. The choice of using a `TreeMap` is given by the easy access to a descending order of the keys. In our case it helps with iterating in a descending exponent order through the polynomial. Also, the access to the last key helps with the division when we need to get the Leading Term.

PUT/GET worst case complexities: $O(\log n)$

Interface:

Validator: contains one method `validate()` which is currently used in the `InputStringValidator` class for checking if the introduced strings have the right format.
[Reference 1.1.](#)

Used algorithms:

Long division algorithm:

Step 1: Order the monomials of the two polynomials P and Q in descending order according to their degree. (Already sorted because we are using a `TreeMap`).

Step 2: Divide the polynomial with the highest degree to the other polynomial having a lower degree (consider first polynomial(P) has the highest degree).

Step 3: Divide the first monomial of P to the first monomial of Q and obtain the first term of the quotient.

Step 4: Multiply the quotient with Q and subtract the result from P obtaining the remainder of the division.

Step 5: Repeat the procedure from step 2 considering the remainder as the new dividend of the division, until the degree of the remainder is lower than Q.

GUI Design:

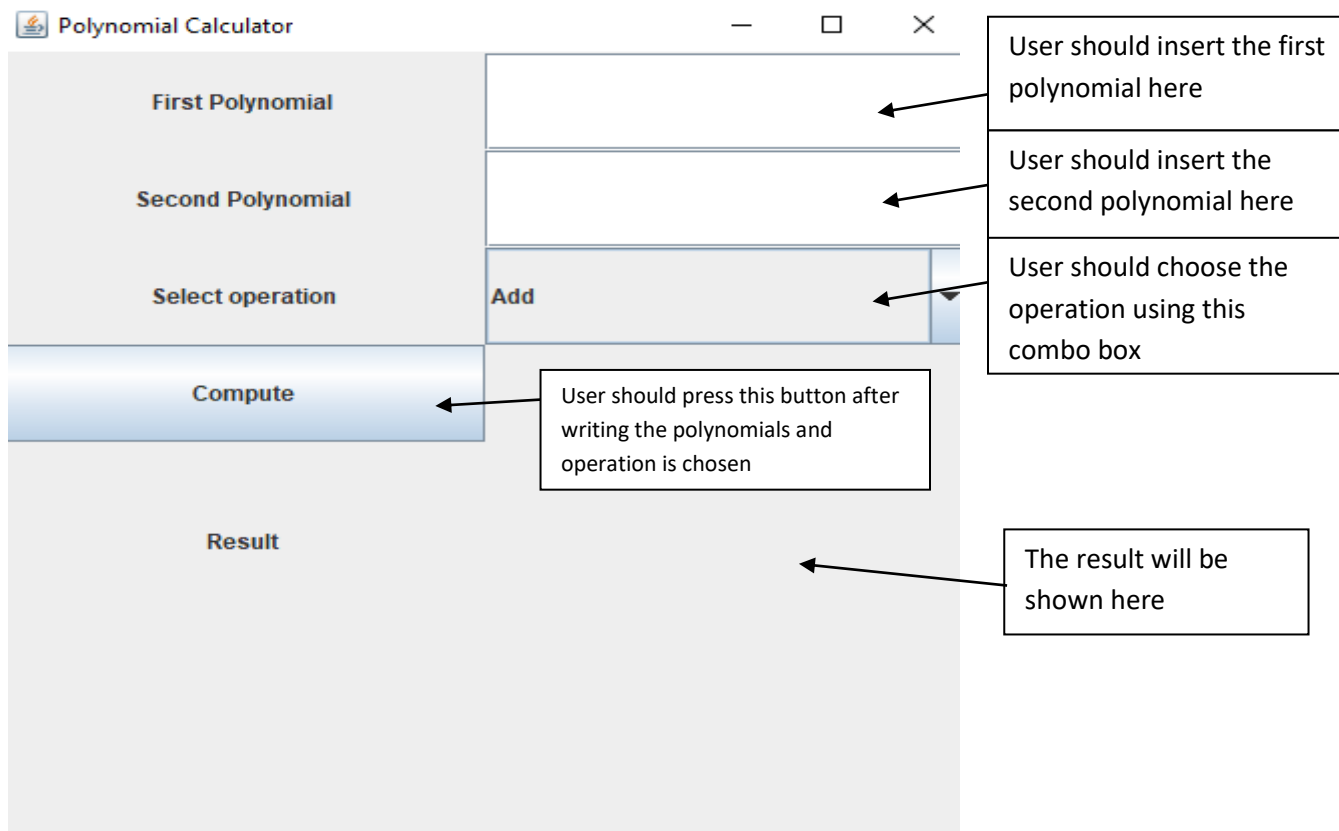
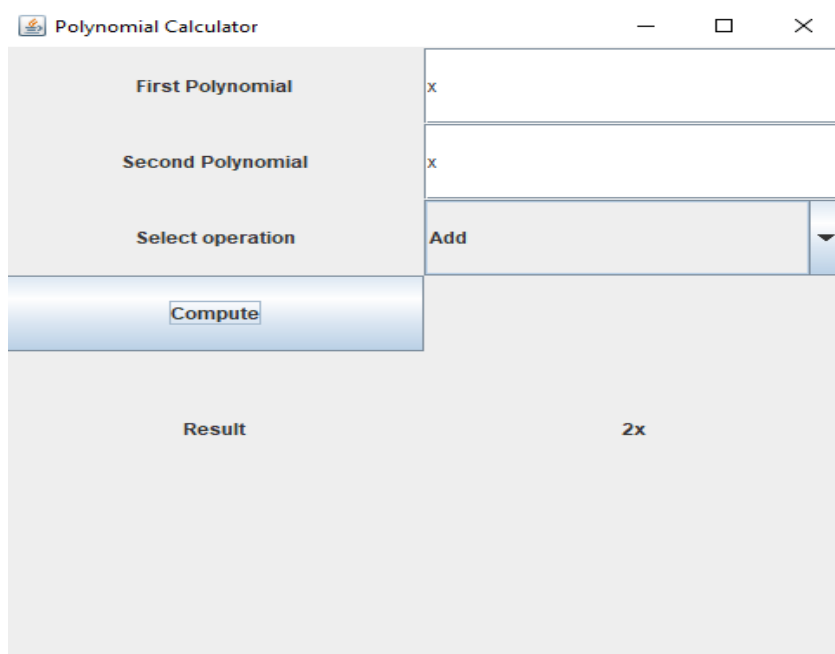
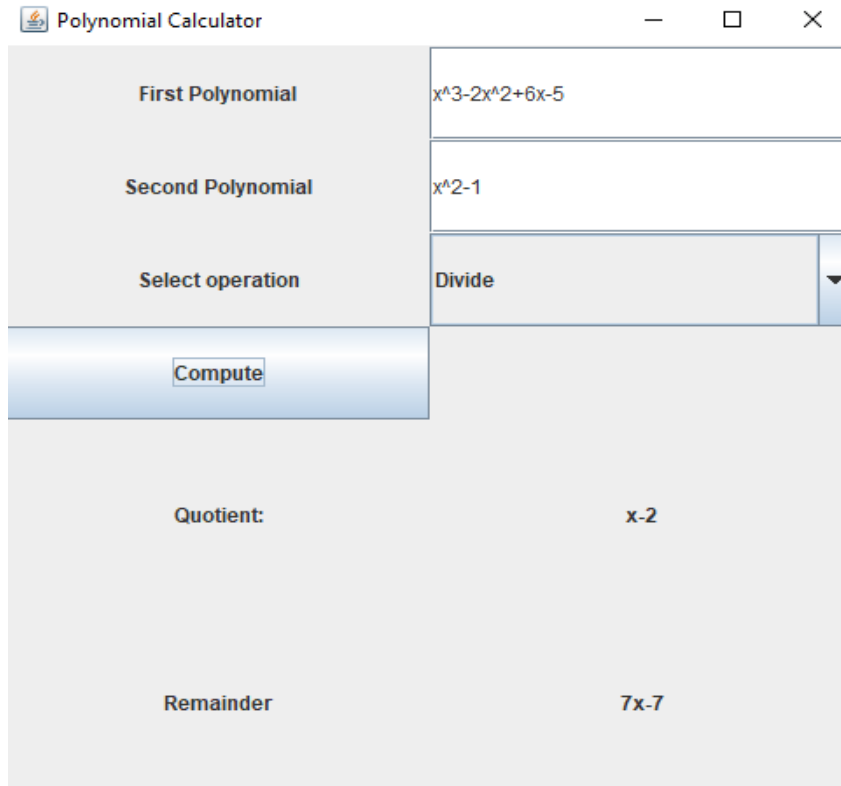


Figure 5. Graphical User Interface

The GUI will look like this in case that Addition, Subtraction, Multiplication, Derivation or Integration are performed:



The GUI will look like this in case of division:



The image shows a window titled "Polynomial Calculator" with standard window controls (minimize, maximize, close). The interface is divided into several sections. On the left, there are labels for "First Polynomial", "Second Polynomial", "Select operation", and a "Compute" button. On the right, there are input fields for the polynomials and a dropdown menu for the operation. Below the input fields, the results for "Quotient" and "Remainder" are displayed.

First Polynomial	$x^3 - 2x^2 + 6x - 5$
Second Polynomial	$x^2 - 1$
Select operation	Divide ▼
Compute	
Quotient:	$x - 2$
Remainder	$7x - 7$

There can appear two types of errors: Wrong Polynomial Syntax and Division by Zero is Illegal.

Invalid Input: appears when one of the inserted polynomials does not respect the format [Reference 1.1](#).

Polynomial Calculator

First Polynomial

2x^2+4x

Second Polynomial

xxxx

Select operation

Divide

Compute

Quotient:

ERROR:

Wrong Polynomial Syntax

Division by Zero is Illegal: appears when the divisor either is zero, or reduces to 0.

Polynomial Calculator

First Polynomial

2x^2+4x

Second Polynomial

x-x

Select operation

Divide

Compute

Quotient:

ERROR:

Division by Zero is Illegal

4. Implementation

Classes:

View Package classes:

CalculatorView:

Fields: contains all the components of the Frame created for the GUI (Labels, ComboBox, Button, TextFields) and an instance of the controller for controlling the behavior of the application.

Methods: contains the methods used for setting up the GUI:

prepareGui(): creates a grid layout with 2 rows and 2 columns in which the resultPanel and the numbersPanel are placed.

prepareResultPanel(): creates a grid layout with 2 rows and 2 columns, first column is used for labels ("Result", "Remainder", "Error!"). and second column is used for placing the actual result of the calculations or the error message.

prepareNumbersPanel(): creates a grid layout with 4 rows and 2 columns. First row is used for the first polynomial: The label "First polynomial" in first column and a text field for the polynomial input. Second row does the same thing but for the second polynomial. Third row is used for operation selection through the combo box placed in the second column. The last column contains only the compute button.

Controller:

Fields: contains an attribute of type CalculatorView. The view that the controller must handle.

Methods:

actionPerformed(ActionEvent e): - if the "Compute" button is pressed in the GUI, this method tries to validate the inputs introduced, in case of wrong format ([Reference 1.1](#)) the GUI will show an error message. Otherwise, the inputs will be parsed to polynomials. Depending on the operation choice, a different method from the Operations class will be called. After performing the operations, the result will be shown in the GUI.

InputStringValidator:

Fields: contains one static final field which represents the pattern that a string inserted in the GUI must follow to be considered correct. ([Reference 1.1](#))

Methods: the class implements the Validator interface and overrides the validate() method in which the Matcher class is used for checking if the string matches the given pattern.

BusinessLogic Package classes:

Operations:

Fields: no fields.

Methods:

+Add(Polynomial polynomialA, Polynomial polynomialB): this method puts the entire polynomialA in the result, then iterates through the polynomial adding the value to the existing keys in the result and putting the values of polynomial where the key doesn't already exist.

```
public static Polynomial Add( Polynomial polynomialA, Polynomial polynomialB
) {
    Polynomial result = new Polynomial();
    for( Map.Entry<Integer, Number> entry :
polynomialA.getMonomials().entrySet() ) {
        result.getMonomials().put(entry.getKey(), entry.getValue());
    }
    for( Map.Entry<Integer, Number> entry :
polynomialB.getMonomials().entrySet() ) {
        if (result.getMonomials().containsKey(entry.getKey())) {
            result.getMonomials().put(entry.getKey(),
result.getMonomials().get(entry.getKey()).intValue() +
entry.getValue().intValue());
        } else result.getMonomials().put(entry.getKey(), entry.getValue());
        if (result.getMonomials().get(entry.getKey()).floatValue() == 0.0)
            result.getMonomials().remove(entry.getKey());
    }
    return result;
}
```

+Subtract(Polynomial polynomialA, Polynomial polynomialB): works in the same way with addition, but subtracts instead of adding.

```
public static Polynomial Subtract( Polynomial polynomialA, Polynomial
polynomialB ) {
    Polynomial result = new Polynomial();
    for( Map.Entry<Integer, Number> entry :
polynomialA.getMonomials().entrySet() ) {
        result.getMonomials().put(entry.getKey(), entry.getValue());
    }
    for( Map.Entry<Integer, Number> entry :
polynomialB.getMonomials().entrySet() ) {
        if (result.getMonomials().containsKey(entry.getKey())) {
            result.getMonomials().put(entry.getKey(),
result.getMonomials().get(entry.getKey()).floatValue() -
entry.getValue().floatValue());
        } else result.getMonomials().put(entry.getKey(), -
entry.getValue().floatValue());
        if (result.getMonomials().get(entry.getKey()).floatValue() == 0.0)
            result.getMonomials().remove(entry.getKey());
    }
    return result;
}
```

+Multiply(Polynomial polynomialA, Polynomial polynomialB): takes each entry from the Map corresponding to polynomialA(each term of polynomialA) and multiplies it with every entry corresponding to polynomialB(each term of polynomialB) and places the result in the result. If the key already exists, the result is added to the already existing value.

```
public static Polynomial Multiply( Polynomial polynomialA, Polynomial
polynomialB ) {
    Polynomial result = new Polynomial();
    for( Map.Entry<Integer, Number> entryA :
polynomialA.getMonomials().descendingMap().entrySet() ) {
        for( Map.Entry<Integer, Number> entryB :
polynomialB.getMonomials().descendingMap().entrySet() ) {
            if (!result.getMonomials().containsKey(entryA.getKey() +
entryB.getKey())) {
                result.getMonomials().put(entryA.getKey() + entryB.getKey(),
entryA.getValue().floatValue() * entryB.getValue().floatValue());
            } else {
                result.getMonomials().put(entryA.getKey() + entryB.getKey(),
result.getMonomials().get(entryA.getKey() + entryB.getKey()).floatValue() +
entryA.getValue().floatValue() * entryB.getValue().floatValue());
            }
        }
    }
    return result;
}
```

+Divide(Polynomial polynomialA, Polynomial polynomial): uses the Long Division Algorithm ([Reference 3.1](#)). It returns an array of length 2 in which the first position contains the quotient and the second position contains the remainder. Each time we create a new monomial to be able to divide the dividend to only the monomial.

```
public static Polynomial[] Divide( Polynomial polynomialA, Polynomial
polynomialB ) {
    Polynomial[] result = new Polynomial[2];
    result[0] = new Polynomial();
    result[1] = new Polynomial();
    if (!polynomialB.getMonomials().isEmpty()){
        while (!polynomialA.getMonomials().isEmpty() &&
polynomialA.getMonomials().lastKey() >= polynomialB.getMonomials().lastKey())
        {
            result[0].getMonomials().put(polynomialA.getMonomials().lastKey()
- polynomialB.getMonomials().lastKey(),
polynomialA.getMonomials().get(polynomialA.getMonomials().lastKey()).floatVal
ue() /
polynomialB.getMonomials().get(polynomialB.getMonomials().lastKey()).floatVal
ue());
            polynomialA = Operations.Subtract(polynomialA,
Operations.Multiply(new Polynomial(polynomialA.getMonomials().lastKey() -
polynomialB.getMonomials().lastKey(),
polynomialA.getMonomials().get(polynomialA.getMonomials().lastKey()).floatVal
ue() /
polynomialB.getMonomials().get(polynomialB.getMonomials().lastKey()).floatVal
```

```

ue()), polynomialB));
    }
    result[1] = polynomialA;}
    else throw new ArithmeticException("Division by Zero");
    return result;
}

```

+Derive(Polynomial polynomial): It iterates through the polynomial and places in the result at the (entry key – 1) the value of (entry coefficient * entry key).

```

public static Polynomial Derive( Polynomial polynomial ) {
    Polynomial result = new Polynomial();
    for( Map.Entry<Integer, Number> entry :
polynomial.getMonomials().entrySet() ) {
        if (entry.getKey() != 0) {
            result.getMonomials().put(entry.getKey() - 1, entry.getKey() *
entry.getValue().intValue());
        }
    }
    return result;
}

```

+Integrate(Polynomial polynomial): It iterates through the polynomial and places in the result at the (entry key + 1) the value of (entry coefficient / entry key + 1);

```

public static Polynomial Integrate( Polynomial polynomial ) {
    Polynomial result = new Polynomial();
    for( Map.Entry<Integer, Number> entry :
polynomial.getMonomials().descendingMap().entrySet() ) {
        result.getMonomials().put(entry.getKey() + 1,
entry.getValue().floatValue() / (entry.getKey() + 1));
    }
    return result;
}

```

Model Package classes:

Polynomial:

Fields:

TreeMap<Integer,Number> monomials: it contains the coefficient and exponent of each monomial that composes the polynomial.

Methods:

toString(Polynomial polynomial): it takes the polynomial and turns into a string that has the desired print format [Reference 1.1](#).

Utils Package classes:

PolynomialParser:

Fields: it contains a static final Pattern field which is grouped in 3 groups in order to be able to get the coefficient and power using the group() method of Matcher class.

Methods:

+parsePolynomial(): whenever the matcher finds a match, it takes the 3 groups separately, first group is the coefficient of that monomial, the second group is the variable “x” and the power sign if it exists and the third group is the exponent. The method takes the first and third group and puts them into the TreeMap corresponding to the Polynomial.

StringNumberParser:

Fields: no fields:

Methods:

parseInt(Number number): It takes a number and if it is parsable to Integer it returns a string with an Integer format, otherwise a string with float format.

5. Results

Testing can be realized directly through the GUI, by checking known polynomial operations. Tests on this project implementing a polynomial calculator has been done using JUnit testing. JUnit is a unit testing open-source framework for the Java programming language. Java Developers use this framework to write and execute automated tests. In Java, there are test cases that have to be re-executed every time a new code is added. This is done to make sure that nothing in the code is broken

Addition:

```
@Test
public void addTest() {
    Polynomial polynomialA, polynomialB, result;
    polynomialA = new Polynomial();
    polynomialA.getMonomials().put(5, 4);
    polynomialA.getMonomials().put(4, -3);
    polynomialA.getMonomials().put(2, 1);
    polynomialA.getMonomials().put(1, -8);
    polynomialA.getMonomials().put(0, 1);
    polynomialB = new Polynomial();
    polynomialB.getMonomials().put(4, 3);
    polynomialB.getMonomials().put(3, -1);
    polynomialB.getMonomials().put(2, 1);
    polynomialB.getMonomials().put(1, 2);
    polynomialB.getMonomials().put(0, -1);
    result = new Polynomial();
    result.getMonomials().put(5, 4);
    result.getMonomials().put(3, -1);
    result.getMonomials().put(2, 2);
    result.getMonomials().put(1, -6);

    assertEquals(Operations.Add(polynomialA, polynomialB).toString(), result.toString());
}
```

$$4 * X^5 - 3 * X^4 + X^2 - 8 * X + 1 +$$
$$3 * X^4 - X^3 + X^2 + 2 * X - 1 =$$
$$4 * X^5 - X^3 + 2 * X^2 - 6 * X$$

✓ testAdd

6 ms

Subtraction:

```
@Test
public void testSubtract() {
```

```

    Polynomial polynomialA, polynomialB, result;
    polynomialA = new Polynomial();
    polynomialA.getMonomials().put(5, 4);
    polynomialA.getMonomials().put(4, -3);
    polynomialA.getMonomials().put(2, 1);
    polynomialA.getMonomials().put(1, -8);
    polynomialA.getMonomials().put(0, 1);
    polynomialB = new Polynomial();
    polynomialB.getMonomials().put(4, 3);
    polynomialB.getMonomials().put(3, -1);
    polynomialB.getMonomials().put(2, 1);
    polynomialB.getMonomials().put(1, 2);
    polynomialB.getMonomials().put(0, -1);
    result = new Polynomial();
    result.getMonomials().put(5, 4);
    result.getMonomials().put(4, -6);
    result.getMonomials().put(3, 1);
    result.getMonomials().put(1, -10);
    result.getMonomials().put(0, 2);

    assertEquals(Operations.Subtract(polynomialA, polynomialB).toString(), result.toString());
}

```

$$\begin{aligned}
 &4 * X^5 - 3 * X^4 + X^2 - 8 * X + 1 - \\
 &3 * X^4 - X^3 + X^2 + 2 * X - 1 = \\
 &4 * X^5 - 6 * X^4 + X^3 - 10 * X + 2
 \end{aligned}$$

✓ testSubtract

0ms

Multiply:

```

@Test
public void testMultiply() {
    Polynomial polynomialA, polynomialB, result;
    polynomialA = new Polynomial();
    polynomialA.getMonomials().put(2, 3);
    polynomialA.getMonomials().put(1, -1);
    polynomialA.getMonomials().put(0, 1);
    polynomialB = new Polynomial();
    polynomialB.getMonomials().put(1, 1);
    polynomialB.getMonomials().put(0, -2);
    result = new Polynomial();
    result.getMonomials().put(3, 3);
    result.getMonomials().put(2, -7);
    result.getMonomials().put(1, 3);
    result.getMonomials().put(0, -2);

    assertEquals(Operations.Multiply(polynomialA, polynomialB).toString(), result.toString());
}

```

$$\begin{aligned}
 &3 * X^2 - X + 1 * \\
 &X - 2 = \\
 &3 * X^3 - 7 * X^2 + 3 * X - 2
 \end{aligned}$$

✓ testMultiply

1 ms

Division:

```
@Test
public void testDivision() {
    Polynomial polynomialA, polynomialB;
    Polynomial[] result = new Polynomial[2];
    polynomialA = new Polynomial();
    polynomialA.getMonomials().put(3, 1);
    polynomialA.getMonomials().put(2, -2);
    polynomialA.getMonomials().put(1, 6);
    polynomialA.getMonomials().put(0, -5);
    polynomialB = new Polynomial();
    polynomialB.getMonomials().put(2, 1);
    polynomialB.getMonomials().put(0, -1);
    result[0] = new Polynomial();
    result[0].getMonomials().put(1, 1);
    result[0].getMonomials().put(0, -2);
    result[1] = new Polynomial();
    result[1].getMonomials().put(1, 7);
    result[1].getMonomials().put(0, -7);
    Polynomial[] forCheck;
    forCheck = Operations.Divide(polynomialA, polynomialB);
    assertEquals(forCheck[0].toString(), result[0].toString());
    assertEquals(forCheck[1].toString(), result[1].toString());
}
```

$$X^3 - 2 * X^2 + 6 * X - 5 /$$

$$X^2 - 1 =$$

$$Q: X - 2$$

$$R: 7X - 7$$

✓ testDivision

0 ms

Derivation:

```
@Test
public void testDerive() {
    Polynomial polynomial, result;
    polynomial = new Polynomial();
    polynomial.getMonomials().put(3, 1);
    polynomial.getMonomials().put(2, -2);
    polynomial.getMonomials().put(1, 6);
    polynomial.getMonomials().put(0, -5);
    result = new Polynomial();
    result.getMonomials().put(2, 3);
    result.getMonomials().put(1, -4);
    result.getMonomials().put(0, 6);
    assertEquals(Operations.Derive(polynomial).toString(), result.toString());
}
```

$$d/dx(X^3 - 2 * X^2 + 6 * X - 5) = 3 * X^2 - 4 * X + 6$$

✓ testDerive

0 ms

Integration:

```
@Test
public void testIntegrate() {
    Polynomial polynomial, result;
    polynomial = new Polynomial();
    polynomial.getMonomials().put(3, 1);
    polynomial.getMonomials().put(2, 4);
    polynomial.getMonomials().put(0, 5);
    result = new Polynomial();
    result.getMonomials().put(4, 0.25);
    result.getMonomials().put(3, 1.3333334);
    result.getMonomials().put(1, 5);

    assertEquals(Operations.Integrate(polynomial).toString(), result.toString());
}
```

Integral($X^3 + 4x^2 + 5$) = $\frac{1}{4}x^4 + \frac{4}{3}x^3 + 5x$

✓ testIntegrate

0 ms

✓ Tests passed: 6 of 6 tests - 7 ms

6. Conclusions

In conclusion, the Polynomial Calculator has been implemented using a TreeMap based approach in which the exponent is the key and the coefficient is the value. The Graphical User Interface was implemented using Java Swing.

Working on this project has helped me to develop a more structured mode of working because I had my code split into packages containing only classes from the same domain of work and also I have learnt to better structure my classes, in such a way that they respect SRR(Single Responsibility Rule). Also, the usage of static methods has become more clearer. I have also learned how to use Java Swing for creating Graphical User Interfaces. A new concept introduced was the testing with JUnit which helped because I didn't have to write the Polynomials each time for testing. I also got my Regular Expressions knowledge on another level.

Future developments can consist into a better looking GUI and solving of other problems like finding the roots of a polynomial equation.

7. Bibliography

- [1] <https://dsrl.eu/courses/pt/> -> PT_2024_A1_S1, PT_2024_A1_S2, PT_2024_A1_S3, Java Swing, Lectures
- [2] <https://junit.org/junit5/docs/current/user-guide/>
- [3] <https://courses.lumenlearning.com/waymakercollegealgebra/chapter/polynomial-long-division/>