

DOCUMENTATION

ASSIGNMENT 1

STUDENT NAME: Balas Andrei-Ioan
GROUP:30424

CONTENTS

1. Assignment Objective.....	3
2. Problem Analysis, Modeling, Scenarios, Use Cases	3
3. Design	5
4. Implementation.....	6
5. Results.....	15
6. Conclusions	18
7. Bibliography.....	18

1. Assignment Objective

(I) *The main objective*

The main objective of the assignment is to create an application that implements polynomial procedures. The operations that this project contains are polynomial addition, subtraction, multiplication, division, derivation, and integration. This leads to the combination of mathematics and programming at the same time.

A polynomial is a mathematical expression consisting of monomials (ex: $2x^3$ is a monomial) that are combined using addition, subtraction, and multiplication. The variables are represented by letters, usually x or y .

General form of a polynomial is $P(x) = a_n * x^n + a_{n-1} * x^{n-1} + \dots + a_1 * x + a_0$

(II) *The sub-objectives*

The sub-objectives are to combine programming with mathematics and at the same time to learn about Java concepts, data structures, and mathematical algorithms like long division of polynomials.

First, the polynomials need a data structure to be stored, so we have to pick one that suits the best. The map, treemap, will be the best in this case because each power of the polynomial will represent the key and each coefficient the value, this will give an advantage over other data structures like ArrayList for example. Treemap also stores the keys in ascending order, this will help us in division operation.

Next, I divided the program into packages and classes. The data.models package contains the polynomial class which stores the data structure. The business.logic package contains the operations class. The gui package contains the HashMapTransform class and Interface class. The org.example contains the App class where the main function is.

In the third step, I implemented the algorithms on the polynomials, besides the ones that I mentioned above I created an algorithm that transforms the polynomial into a string, and also another one that those the opposite.

The fourth step was to test the algorithms in a JUnit class.

The final step that led to the main objective was to connect the front end (the user interface) with the back end (the operations).

2. Problem Analysis, Modeling, Scenarios, Use Cases

(I) *Problem Analysis*

The application should perform the operations mentioned above using an interface that takes two polynomials inserted from the keyboard by the user. Using an interface implies that we use the abstraction principle in which we hide from the user

unnecessary information like the classes, the packages, and the methods. The result of the procedures will only display the result if the form of the polynomials is correct, else the user needs to insert again the polynomials, so we need to make a pattern check for each field.

(II) *Modeling*

Addition, subtraction, multiplication, and division will perform on both polynomials. The derivation and integration will perform on only one polynomial, so we choose the first polynomial. So, we put six buttons for each operation performed and two editable text fields two insert each polynomial, and two non-editable text fields, one for the result and one for the remainder, in case we have one when we divide.

(III) *Scenarios*

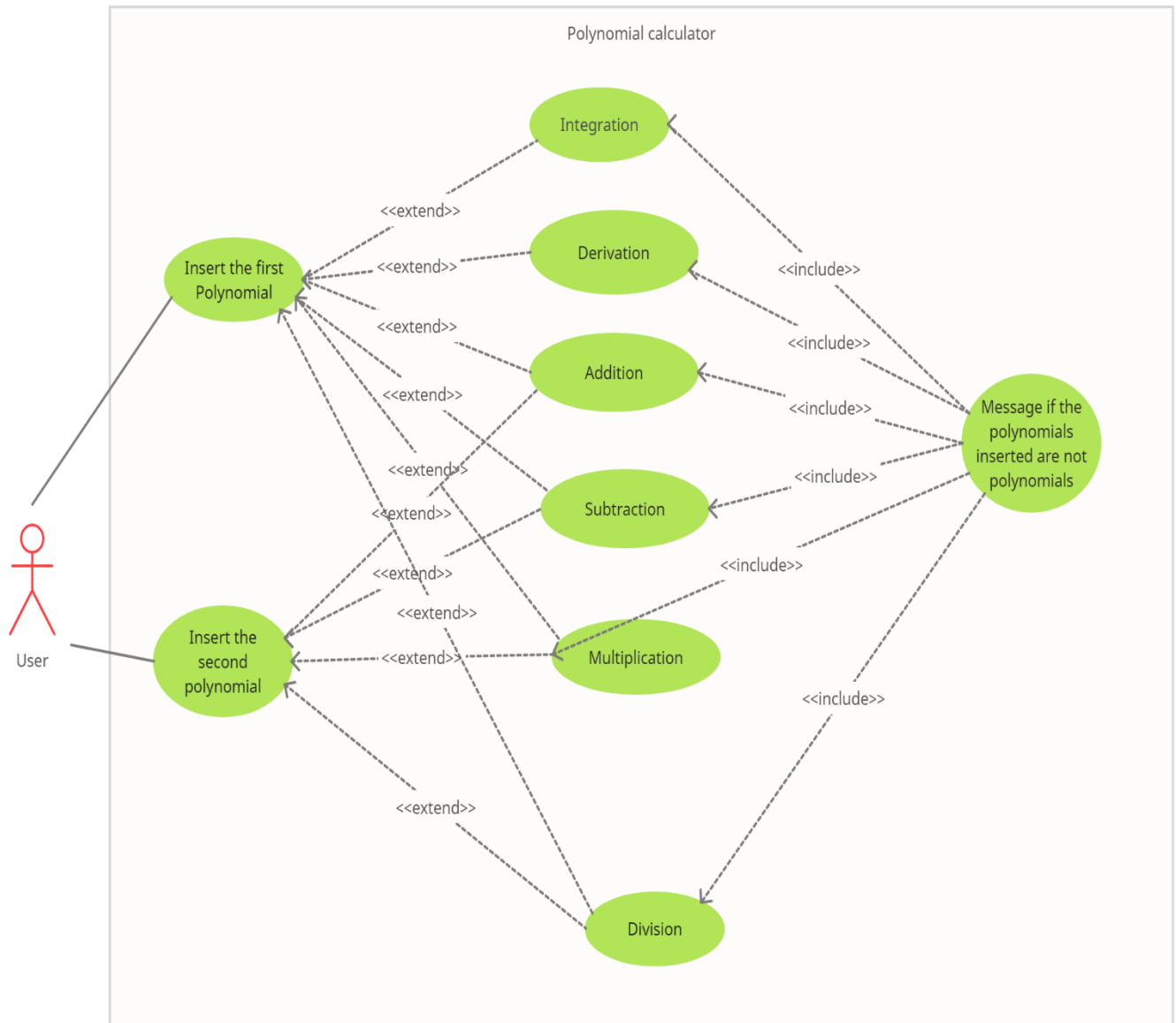
Different scenarios where the user doesn't introduce the polynomials correctly need to be treated in the back-end part.

In case the user presses the addition, subtraction, multiplication, or division button we first need to check if both polynomials were inserted, if not a pop-up with an error message will be displayed. Next, we need to check if the first polynomial and the second one are inserted correctly, else another pop-up will appear which tells the user the polynomial that isn't valid.

For the derivation and integration, we check only the first polynomial. If the operation is performed the application will display a pop-up with a warning message that tells the user that the operation will work only on the first one.

To make it more user-friendly, the polynomials can be introduced also with spaces between variables but also without them. For example: " x^2+2x+1 " and " $x^2 + 2x + 1$ " will be the same and valid.

(IV) *Use cases*



3. Design

The main object in the application is the polynomial.

The polynomial class contains the following attributes and methods:

- Attributes: the TreeMap data structure in which the monomials are stored.

```
TreeMap<Integer, Double> monomials = new
TreeMap<Integer, Double>();
```

- Methods: -the getmonomial() is a getter that return the monomials map

```
public TreeMap<Integer, Double> getMonomials() {
    return monomials;
}
```

- print(), prints the map

```
public void print() {  
    System.out.println(monomials);  
}
```

-leadingCoefficient() returns the coefficient of the term
monomial with the biggest degree

```
public Double leadingCoefficient() {  
    return monomials.get(monomials.lastKey());  
}
```

Another important class is the Operations class which is made only of methods
that are static and return a polynomial object:

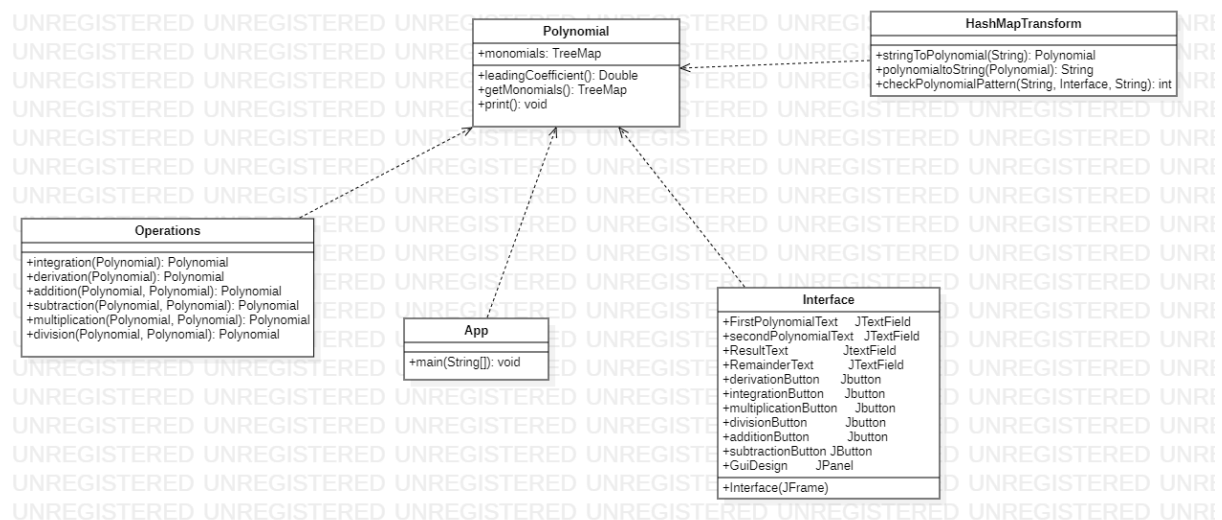
-addition

-subtraction

-multiplication

-division

The UML diagram:



4. Implementation

(I) Polynomial class

(II) *Operations class*

```
        public static Polynomial addition(Polynomial one, Polynomial two) {
            Polynomial res = new Polynomial();

            //for each exponent from the first polynomials we add the coefficient
            //from the another polynomial if the exponent exists else we assume it's 0
            one.getMonomials().forEach((exponent, coefficient)->{
                double
                finalcoefficient=coefficient+two.getMonomials().getOrDefault(exponent,0.00)
                ;
                if(finalcoefficient!=0)

            res.getMonomials().put(exponent,coefficient+two.getMonomials().getOrDefault
            (exponent,0.00));
            });

            //we need to put the rest of the exponents that exist in second
            polynomial but does not exist in the first one
            two.getMonomials().forEach((exponent,coefficient)->{
                if(!one.getMonomials().containsKey(exponent) && coefficient!=0){
                    res.getMonomials().put(exponent,coefficient);
                }
            });
            return res; //return the result
        }

        public static Polynomial subtraction(Polynomial one, Polynomial two){
            Polynomial res=new Polynomial();

            //we use the same approach as in the addition, but we need to remove
            the exponents where the result coefficient from the subtraction is 0
            one.getMonomials().forEach((exponent,coeffiecient)->{
                res.getMonomials().put(exponent,coeffiecient-
                two.getMonomials().getOrDefault(exponent,0.00));
                if (coeffiecient-
                two.getMonomials().getOrDefault(exponent,0.00)==0){
                    res.getMonomials().remove(exponent);
                }

            });

            //we need to put the rest of the exponents that exist in second
            polynomial but does not exist in the first one
            two.getMonomials().forEach((exponent,coefficient)->{
                if(!one.getMonomials().containsKey(exponent)){
                    res.getMonomials().put(exponent,-1*coefficient);
                }
            });

            return res; //return the result
        }
    }
```

```
        public static Polynomial multiplication(Polynomial one, Polynomial two){
            Polynomial res=new Polynomial();

            //we multiply each element from the first polynomial with each element
            from the second polynomial
            //we add the exponents then we multiply the coeffiecent, in order to get
            the correct result we need to see if the exponent repeats and if it repeats
        }
```

```

we need to add the current sum of the coefficient
one.getMonomials().forEach((exponentOne,coefficientOne)->{
    two.getMonomials().forEach((exponentTwo,coefficientTwo)->{
        int finalExponent=exponentOne+exponentTwo;
        double finalCoeffiecient=coefficientOne*coefficientTwo;

res.getMonomials().put(finalExponent,res.getMonomials().getOrDefault(finalE
xponent,0.00)+finalCoeffiecient);
    });
});

return res;
}

```

```

public static Polynomial derivation(Polynomial one){
    Polynomial res=new Polynomial();

    //we go through the polynomial. We reduce 1 from each exponent, then
    multiply the coeff with exponent if the exponent it's not 0
    one.getMonomials().forEach((exponent,coefficient)->{
        if(exponent!=0){
            int finalExponent=exponent-1;
            double finalCoeffiecient=coefficient*exponent;
            res.getMonomials().put(finalExponent,finalCoeffiecient);
        }
    });

    return res;
}

```

```

public static Polynomial integration(Polynomial one){
    Polynomial res=new Polynomial();

    //we go through the polynomial. We add 1 to each exponent.We divide the
    coefficient by the exponent if the exponent it's not 0
    one.getMonomials().forEach((exponent,coefficient)->{
        int finalExponent=exponent+1;
        double finalCoeffiecient=1;
        finalCoeffiecient=coefficient/finalExponent;
        res.getMonomials().put(finalExponent,finalCoeffiecient);
    });

    return res;
}

```

```

public static Polynomial[] divide(Polynomial dividend, Polynomial
divisor){
    Polynomial res=new Polynomial();
    Polynomial remainder=dividend;

    //we divide the polynomials while the degree of the remainder is bigger
    than the degree of the divisor
    while(remainder.getMonomials().size()>=divisor.getMonomials().size()){

        //we divide the biggest the term, subtract the powers and divide
        the coefficients
    }
}

```



```

        int degree=remainder.getMonomials().lastKey()-
divisor.getMonomials().lastKey();
        //the coefficient will be the coefficient of the remainder/ the
coefficient of the divisor
        double
coefficient=remainder.leadingCoefficient()/divisor.leadingCoefficient();

        //create a new polynomial in which we will store each monomial
Polynomial term=new Polynomial();
term.getMonomials().put(degree,coefficient);
//add the current monomial from term
res=addition(res,term);
//store dividend from wich we will subtsract the remainder
Polynomial dividend2=remainder;
//get the remainder by multiplying the monomial term obtained from
the division of the leading terms with the divisor
remainder=multiplication(divisor,term);
//subtract to get the new remainder
remainder=subtstraction(dividend2,remainder);

        //do this until the leading term of the divisor is smaller than the
leading term of the remainder
    }
    //return the result
    return new Polynomial[]{res,remainder};
}

```

(III) *HashMapTransform class*

In this class are stored all the methods that operate on the Polynomial

```

public static int checkPolynomialPattern(String toCheck, Interface
parent, String toShow){
    toCheck=toCheck.replaceAll("\\s", "");

    String regexFirstCase="^\\s*([-+]?(\\d*\\.?\\d*)*\\s*x\\s*(\\s*(\\d+\\s*)+|(\\d*\\.?\\d*)*\\s*x?\\s*\\s*+)?\\s*(\\d*\\.?\\d*)*\\s*$"; //if we have more than 2 terms or 1 term in
the polynomial for example x^3+2x+1
    String regexSecondCase="^\\s*([-+]?(\\d*\\.?\\d*)*\\s*x\\s*(\\s*(\\d+\\s*)+|([-+]?\\d*x\\s*)|([-+]?\\d*x(\\s*(\\d+\\s*)+)?\\s*)+|([-+]?\\s*\\d+)|((\\d*\\.?\\d*)*\\s*x?\\s*\\s*+\\s*(\\d*\\.?\\d*)*\\s*$"); //
second case if we have for example -3x+1, 2x, x
    String regexFloatCase= "^([-+]?\\d*\\.?\\d*x\\s*(\\d+\\s*)+|([-+]?\\d*\\.?\\d*x(\\s*(\\d+\\s*)+)?\\s*)+|([-+]?\\s*\\d*\\.?\\d+)?)$";

    if(toCheck.isEmpty()) {
        JOptionPane.showMessageDialog(parent,"Insert " + toShow,"Try
again",JOptionPane.ERROR_MESSAGE);
        return 0;
    }
    if (!toCheck.matches(regexFirstCase) &&
!toCheck.matches(regexSecondCase) && !toCheck.matches(regexFloatCase) ){
        JOptionPane.showMessageDialog(parent,toShow + " isn't correct.
Insert a polynomial","Try again",JOptionPane.ERROR_MESSAGE);
        return 0;
    }
}

```

```

    return 1;
}

```

This is the method that checks if the String input matches the pattern of a polynomial. First, we remove the spaces. I created 3 regex cases. First one is for the case if we have a polynomial of form “ ax^b+cx+d ”, second is for cases we have “ ax^b ”, “ ax ” or a constant. Third one is to assure that we can introduce float coefficients.

If the String is empty it means that the polynomial and we return an error using JOptionPane. If the String does not match any of the patterns we return another JOptionPane error.

```

public static Polynomial stringToPolynomial(String toConvert) {

    Polynomial result = new Polynomial();
    //replace all the spaces if we have spaces between characters
    toConvert=toConvert.replaceAll("\\s", "");
    //replace - with +- to split by +
    toConvert=toConvert.replaceAll("-", "+-");

    String[] terms=toConvert.split("\\+");

    for(String term: terms) {
        //if we have the index ^ means we are on the case we have ax^b
        if (!term.isEmpty()) {
            if (term.indexOf("^") >= 1) {
                double coefficient;
                //split the term by ^
                String[] parts = term.split("x\\^");
                //check if we don't have the coefficient, if it's true we
                //put 1 else we put the coefficient
                if (parts[0].isEmpty())
                    coefficient = 1d;
                else {
                    if(parts[0].equals("-"))
                        coefficient=-1d;
                    else
                        coefficient = Double.parseDouble(parts[0]);
                }
                //add the power and the coefficient to the polynomial
                result.getMonomials().put(Integer.parseInt(parts[1].trim()), coefficient);
            }
            //case if we have only x
            if (term.indexOf("^") == -1 && term.indexOf("x") != -1) {
                double coefficient = 1;
                //if the length of the term is bigger than one it means we
                //have coefficient else we put 1
                if (term.length() > 1) {
                    String[] parts = term.split("x");
                    if (parts[0].isEmpty())
                        coefficient = 1d;
                    else {
                        if(parts[0].equals("-"))
                            coefficient=-1d;

```

```

        else
            coefficient = Double.parseDouble(parts[0]);
        }
    }
    result.getMonomials().put(1, coefficient);
}
//case of the empty term
if (term.indexOf("x") == -1)
    result.getMonomials().put(0, Double.parseDouble(term));
}
}
return result;
}

```

This method converts the input String to Polynomial object. First, we eliminate all spaces, then replace “-” with “+” to split after “+” into tokens. Iterate through the tokens and take the case when we have “ax^b”, split after “^” and put the coefficient and the power into a polynomial. Second case, “ax”, split after “x” and put the coefficient and the power will be 1. The third case is for the constant we put the power as 0 and the coefficient.

```

public static String polynomialToString(Polynomial polynomial) {
    StringBuilder toBuild=new StringBuilder();
    String result;
    boolean isFirstTerm=true;
    if(polynomial.getMonomials().isEmpty())
    {
        toBuild.append("0");
    }
    else {
        for (int power : polynomial.getMonomials().descendingKeySet())
        {
            double coefficient = polynomial.getMonomials().get(power);
            if (coefficient != 0) {
                //if it's not the first term
                if (!isFirstTerm) {
                    if (coefficient > 0) {
                        toBuild.append("+");
                    } else {
                        toBuild.append("-");
                        coefficient = (-1)*coefficient;
                    }
                }
                //if it's the first term

                else {
                    if (coefficient < 0) {
                        toBuild.append("-");
                        coefficient = (-1)*coefficient;
                    }
                    isFirstTerm = false;
                }
                if (coefficient != 1 || power == 0) {
                    toBuild.append(coefficient);
                }
                if (power > 0) {
                    toBuild.append("x");
                    if (power > 1) {
                        toBuild.append("^").append(power);
                    }
                }
            }
        }
    }
}

```


```

    }
    }
    }
    }
    result=toBuild.toString();
    return result;
}
}

```

This method takes a polynomial object and converts it into a String. We iterate through the polynomial from the last term to put the maximum degree as the first term in the string, we test if the coefficient is positive or negative, if it's positive we put + between terms in the String, else we put – and the coefficient is multiplied by -1 to not get a double sign.

(IV) *The user interface*

 Log in
 ×

Polynomial Calculator

First Polynomial

Second Polynomial

Result

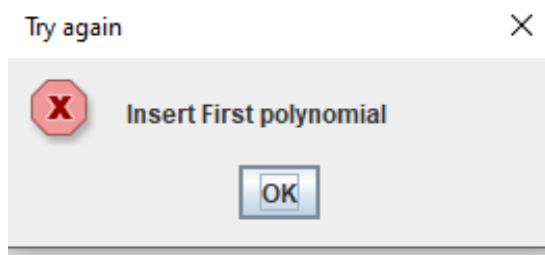
Remainder

Addition	Subtraction
Multiplication	Division
Derivation	Integration

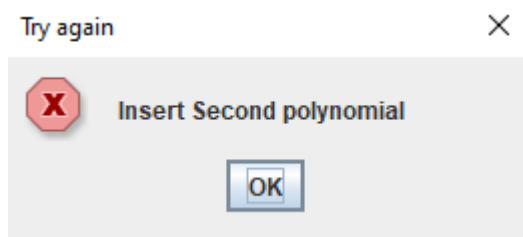
The user interface is made of six buttons: addition, subtraction, multiplication, division, derivation, and integration. Four text fields and for each field a label that name the field.

If we press a button and we do not insert a polynomial in the text fields the pop will be

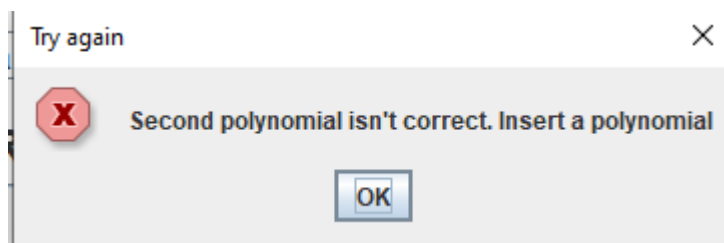
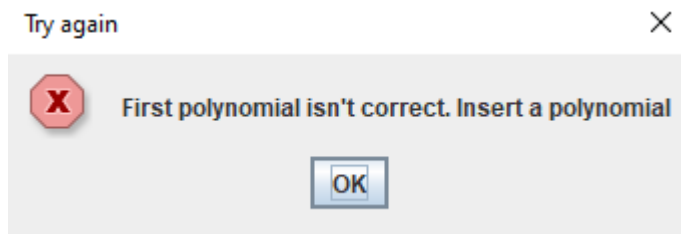
For first polynomial:



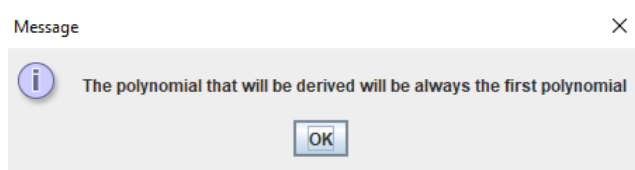
For the second polynomial:



If the polynomial isn't correct

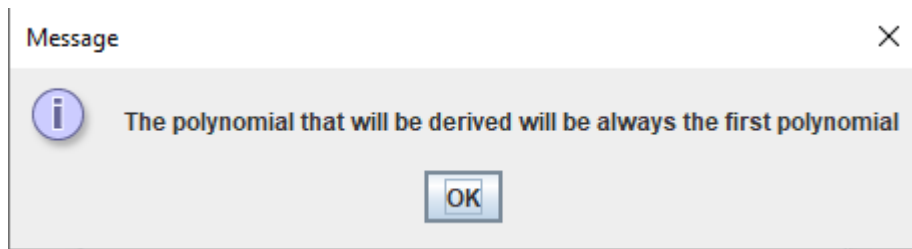


For the derivation, the next message will appear if the first polynomial is correct.

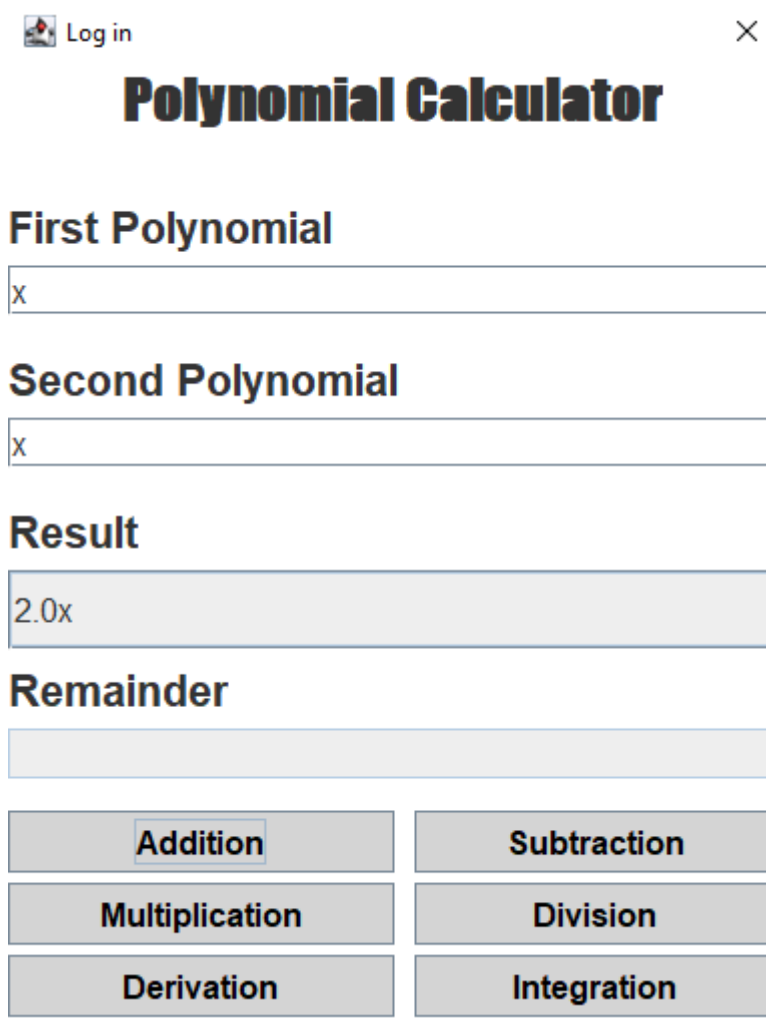


This message will only appear when we derivate first, for the next time we press derivate will not appear.

The same goes for integration.



The result will be displayed like this:



Log in

Polynomial Calculator

First Polynomial

x

Second Polynomial

x


Result

2.0x

Remainder

Addition	Subtraction
Multiplication	Division
Derivation	Integration

The remainder:

 Log in
 ×

Polynomial Calculator

First Polynomial

Second Polynomial

Result

Remainder

Addition	Subtraction
Multiplication	Division
Derivation	Integration

The next time we do another operation the remainder will be cleared.

5. Results

For testing the operations I used JUnit

The results are the following.

```
//addition
Polynomial addition1 =new Polynomial();           //-x^3+17/9x^2+2
Polynomial addition2 =new Polynomial();           //4x^2+1/6
addition1.getMonomials().put(3,-1d);
addition1.getMonomials().put(2, (double) (17/9));
addition1.getMonomials().put(0,2d);

addition2.getMonomials().put(2,4d);
addition2.getMonomials().put(0, (double) (1/6));
Polynomial resultTrue=addition(addition1,addition2);           //-
x^3+53/9x^2+13/6

Polynomial resultExpected=new Polynomial();
resultExpected.getMonomials().put(3,-1d);
resultExpected.getMonomials().put(2, (double) (53/9));
resultExpected.getMonomials().put(0, (double) (13/6));
```

```
String resultT=polynomialtoString(resultTrue);
String resultE=polynomialtoString(resultExpected);
assertEquals(resultE,resultT);
```

✓	✓ Test Results	10 ms
✓	org.example.AppTest	10 ms
✓	testApp	10 ms

```
Polynomial subtraction1=new Polynomial(); //x^2+3x-33
Polynomial subtraction2=new Polynomial(); //x^2+5x+1
subtraction1.getMonomials().put(2,1d);
subtraction1.getMonomials().put(1,3d);
subtraction1.getMonomials().put(0,-33d);

subtraction2.getMonomials().put(2,1d);
subtraction2.getMonomials().put(1,5d);
subtraction2.getMonomials().put(0,1d);

Polynomial resultTtrueS=subtstraction(subtraction1,subtraction2); //-2x-32

String resutltTS=polynomialtoString(resultTtrueS);

Polynomial resultExpectedS=new Polynomial();
resultExpectedS.getMonomials().put(1,-2d);
resultExpectedS.getMonomials().put(0,-34d);
String resultES=polynomialtoString(resultExpectedS);

assertEquals(resultES,resutltTS);
```

✓	✓ Test Results	10 ms
✓	org.example.Ap	10 ms
✓	testApp	10 ms

```
Polynomial multiplication1=new Polynomial();
Polynomial multiplication2=new Polynomial();

multiplication1.getMonomials().put(1,1d);
multiplication1.getMonomials().put(0,2d);

multiplication2.getMonomials().put(1,1d);
multiplication2.getMonomials().put(0,2d);

Polynomial resultTrueM=multiplication(multiplication1,multiplication2);
String resultTM=polynomialtoString(resultTrueM);

Polynomial resultExpcetedM=new Polynomial();
resultExpcetedM.getMonomials().put(2,1d);
resultExpcetedM.getMonomials().put(1,4d);
resultExpcetedM.getMonomials().put(0,4d);

String resultEM=polynomialtoString(resultExpcetedM);
assertEquals(resultEM,resultTM);
```

✓	✓ Test Results	10 ms
✓	org.example.Ap	10 ms
✓	testApp	10 ms


```
//division
Polynomial division1=new Polynomial();
Polynomial division2=new Polynomial();

division1.getMonomials().put(2,1d);
division1.getMonomials().put(1,5d);
division1.getMonomials().put(0,6d);

division2.getMonomials().put(1,1d);
division2.getMonomials().put(0,2d);

Polynomial[] resultTrueD=divide(division1,division2);
Polynomial resultD=resultTrueD[0];
String resultTD=polynomialtoString(resultD);

Polynomial resultExpectedD=new Polynomial();
resultExpectedD.getMonomials().put(1,1d);
resultExpectedD.getMonomials().put(0,3d);

String resultED=polynomialtoString(resultExpectedD);

assertEquals(resultED,resultTD);
```

✓ Test Results	10 ms
✓ org.example.Ap	10 ms
✓ testApp	10 ms

```
Polynomial derivation1=new Polynomial(); //x^2+5x+6
derivation1.getMonomials().put(2,1d);
derivation1.getMonomials().put(1,5d);
derivation1.getMonomials().put(0,6d);

Polynomial resultTrueDe=derivation(derivation1);
String resultTDe=polynomialtoString(resultTrueDe);

Polynomial resultExpectedDe=new Polynomial();
resultExpectedDe.getMonomials().put(1,2d);
resultExpectedDe.getMonomials().put(0,5d);
String resultEDe=polynomialtoString(resultExpectedDe);

assertEquals(resultEDe,resultTDe);

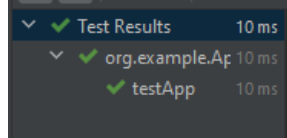
Polynomial integration1=new Polynomial(); //x^2+5x+6
integration1.getMonomials().put(2,1d);
integration1.getMonomials().put(1,5d);
integration1.getMonomials().put(0,6d);
```

✓ Test Results	10 ms
✓ org.example.Ap	10 ms
✓ testApp	10 ms

```
Polynomial resultTrueI=integration(integration1);
String resultTI=polynomialtoString(resultTrueI);

Polynomial resultExpectedI=new Polynomial();
resultExpectedI.getMonomials().put(3,1/3d);
resultExpectedI.getMonomials().put(2,5/2d);
resultExpectedI.getMonomials().put(1,6d);
String resultEI=polynomialtoString(resultExpectedI);
```

```
assertEquals(resultEI, resultTI);
```



A screenshot of a test runner interface showing a tree of test results. The root node is 'Test Results' with a green checkmark and '10 ms'. It has a child node 'org.example.App' with a green checkmark and '10 ms'. This node has a child node 'testApp' with a green checkmark and '10 ms'.

✓	Test Results	10 ms
✓	org.example.App	10 ms
✓	testApp	10 ms

6. Conclusions

In conclusion, from this assignment, I learned that TreeMap is a really good data structure to store any kind of data in which a value is associated with another value. Also, using TreeMap made the implantation of the polynomial operations much easier especially at the division operation, having always access to the last key which stores the biggest degree was very efficient and also because of the forEach function which made the iteration through polynomial much easier. I learned more about pattern matching using regex and StringBuilder.

Some future developments could be to calculate the roots of the polynomials or to show the function on a graph.

7. Bibliography

[Stack Overflow - Where Developers Learn, Share, & Build Careers](#)

<https://en.wikipedia.org/wiki/Polynomial>