FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 1

POLYNOMIAL CALCULATOR

BÎNDILĂ ADRIAN-IOAN
GROUP 30422

1. Assignment objective

Design and implement a polynomial calculator with a dedicated graphical user interface, through which the user is able to insert the polynomials, select the mathematical operation to be performed on them, then view the result.

Secondary objectives:

- Identify problem requirements (2.Problem Analysis)
- Design the polynomial calculator (3.Design)
- Implement the polynomial calculator (4. Implementation)
- Test the polynomial calculator (5. Results)

2. Problem analysis, modeling, scenarios, use cases

Functional requirements:

- Insert two polynomials, using regular mathematical notation
- Parse the user input from any errors
- Perform mathematical operations, such as addition, subtraction, multiplication, derivative, integral
- Clear the value of the currently selected polynomial
- Clear the value of both polynomials and that of the stored result
- Verify that the inserted polynomial was correctly inserted

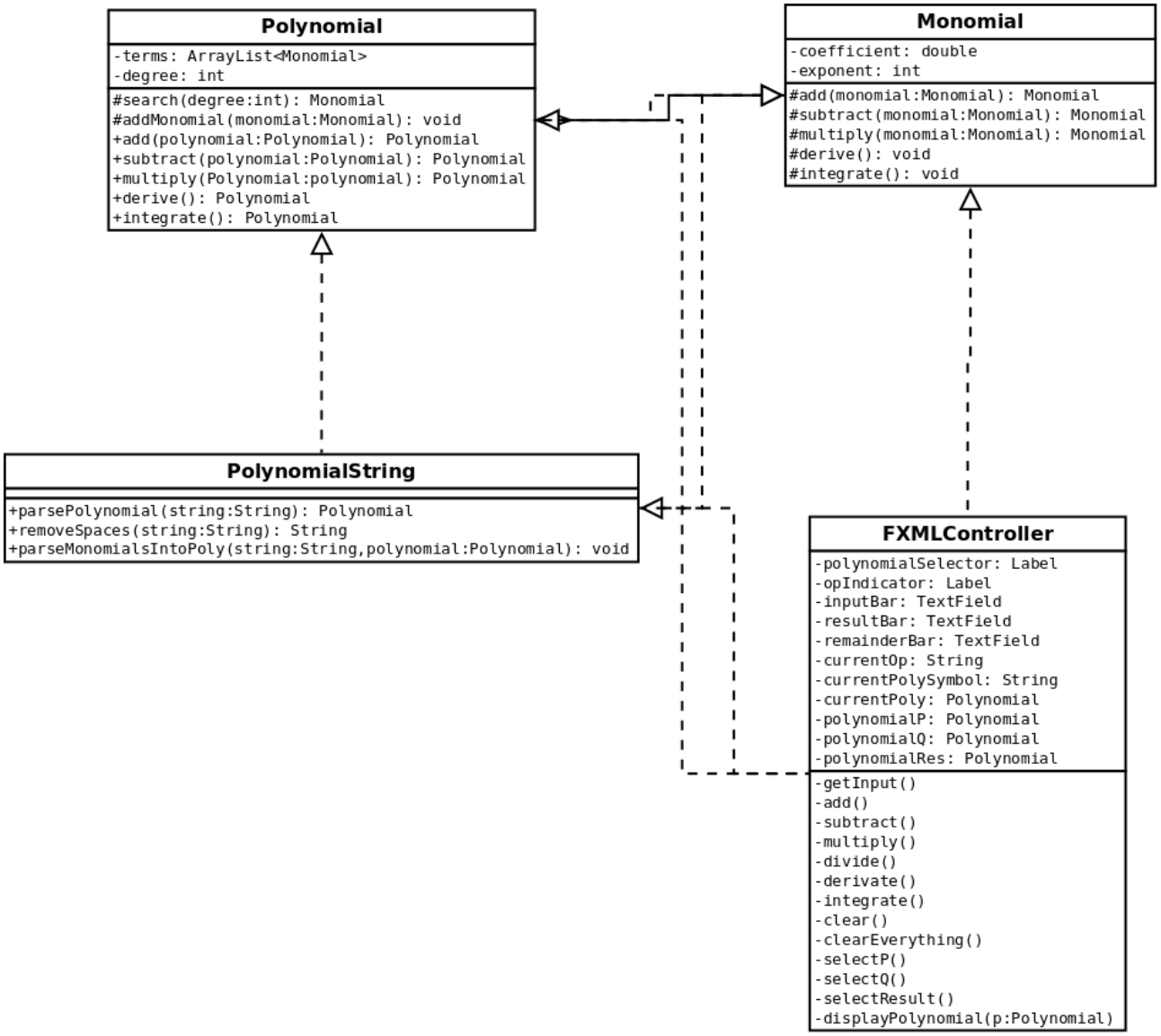
Non-functional requirements:

- Provide an easy to use graphical user interface to facilitate ease of access
- Provide a help page to aid the user in the use of the calculator
- Have a good looking interface to attract the user
- Indicate the current polynomial that is being edited
- Indicate the currently selected operation

Use cases:

- Addition/Subtraction/Multiplication/Division:
  1. User inserts the two polynomials, selecting them via buttons.
  2. The user selects the addition/subtraction/multiplication/division operation.
  3. The results can be viewed by pressing the result button.
- Derivation/Integration:
  1. User inserts the polynomial.
  2. User selects the derivative/integral operation
  3. User can view the results via the result button
- Wrong input:
  1. The inputted text can be checked with the polynomial created by the computer
  2. If the user notices an error, they can press a button to reset the calculator to an initial state.

3. Design (design decisions, UML diagrams, data structures, class design, interfaces, relationships, packages, algorithms, user interfaces)



4. Implementation

PolynomialString class:

```
public class PolynomialString {

    public Polynomial parsePolynomial(String string) {...}

    private String removeSpaces(String string) {...}

    private void parseMonomialsIntoPoly(String string, Polynomial polynomial) {...}
}
```

The class is responsible for parsing all the input from the user into a usable polynomial in order to perform operations on it. The public method parsePolynomial is the one being used by the controller class FXMLController, with the other two private methods being used as helpers.

Method - removeSpaces:

```
private String removeSpaces(String string) {
    final Pattern pattern = Pattern.compile( regex: "\\s*", Pattern.CASE_INSENSITIVE);
    final Matcher matcher = pattern.matcher(string);
    string = matcher.replaceAll( replacement: "");
    return string;
}
```

The removeSpaces method is used in order to remove any unnecessary spaces the user may insert, via the regular expression “\\s”.

Method - parseMonomialsIntoPoly:

```
private void parseMonomialsIntoPoly(String string, Polynomial polynomial) {
    string = removeSpaces(string);
    final String regex = "([+-]?)(\\d*)?([*]?[a-z])?([\\^])?([\\d]*)?";
    final Pattern pattern = Pattern.compile(regex);
    final Matcher matcher = pattern.matcher(string);
    while (matcher.find()) {
        if (!matcher.group().isEmpty() && !matcher.group().isBlank()) {
            Monomial monomial = new Monomial();
            if (matcher.group(2) != null && !matcher.group(2).isEmpty()) {
                monomial.coefficient = Double.parseDouble(matcher.group(2));
            } else {
                monomial.coefficient = 1;
            }
            if (matcher.group(1).contains("-")) {
                monomial.coefficient = -monomial.coefficient;
            }
            if (matcher.group(4) != null && !matcher.group(4).isEmpty()) {
                monomial.exponent = Integer.parseInt(matcher.group(5));
            } else {
                if (matcher.group(3) != null && !matcher.group(3).isEmpty()) {
                    monomial.exponent = 1;
                } else {
                    monomial.exponent = 0;
                }
            }
            polynomial.addMonomial(monomial);
        }
    }
}
```

The parseMonomialsIntoPoly method makes use of the regular expression “([+-]?)(\d*)?([*]?[a-z])?([\^])?([\d]*)?” in order to separate the terms of a given monomial term and parse them accordingly. There are 5 groups in total: one is used in order to separate the sign of the coefficient, then the absolute value of it is taken into the second group. Next, the symbol of the polynomial’s variable (usually x) is separated into group 3. Group 4 contains the exponent symbol ^ if any. Group 5 contains the exponent by itself, in order to be parsed accordingly.

Method - parsePolynomial:

```
public Polynomial parsePolynomial(String string) {
    string = removeSpaces(string);
    final Pattern pattern = Pattern.compile("([+-]?[^-+]+)");
    final Matcher matcher = pattern.matcher(string);
    Polynomial polynomial = new Polynomial();
    while (matcher.find()) {
        for (int i = 1; i <= matcher.groupCount(); i++) {

            parseMonomialsIntoPoly(matcher.group(i), polynomial);
        }
    }
    return polynomial;
}
```

The Method extracts the initial polynomial string and breaks it down into component terms, then sends the substrings to the parseMonomialsIntoPoly monomial. The method makes use of the regex pattern “([+-]?[^-+]+)” . It looks for + , - and ^ signs in order to make the separation term by term, since every term is seperated by a plus or minus sign at least.

Monomial Class

```
public class Monomial {
    private double coefficient;
    private int exponent;

    Monomial() {...}

    Monomial(int exponent, double coefficient) {...}

    public double getCoefficient() { return coefficient; }

    public int getExponent() { return exponent; }

    public void setCoefficient(double coefficient) { this.coefficient = coefficient; }

    public void setExponent(int exponent) { this.exponent = exponent; }


    protected Monomial add(final Monomial monomial) {...}

    protected Monomial subtract(final Monomial monomial) {...}

    protected Monomial multiply(final Monomial monomial) {...}

    protected Monomial divide(final Monomial monomial) {...}

    protected void derive() {...}

    protected void integrate() {...}
}
```

The monomial class is used by the polynomial class in order to perform certain mathematical operations such as addition and subtraction, since they require term-by-term operations.

Method – add

```
Monomial add(final Monomial monomial) {
    Monomial result = new Monomial();
    result.coefficient = this.coefficient + monomial.coefficient;
    result.exponent = this.exponent;
    return result;
}
```

The add method adds two monomial terms returning the result as a separate new term.

Method – subtract

```
Monomial subtract(final Monomial monomial) {
    Monomial result = new Monomial();
    result.coefficient = this.coefficient - monomial.coefficient;
    result.exponent = this.exponent;
    return result;
}
```

The subtract method subtracts two monomial terms returning the result as a separate new term.

Method – multiply

```
Monomial multiply(final Monomial monomial) {
    Monomial result = new Monomial();
    result.coefficient = this.coefficient * monomial.coefficient;
    result.exponent = this.exponent + monomial.exponent;
    return result;
}
```

The multiply method performs the multiplication of two given monomials.

Method – divide

```
protected Monomial divide(final Monomial monomial) {
    Monomial result = new Monomial();
    if (monomial != null && monomial.coefficient != 0) {
        result.coefficient = this.coefficient / monomial.coefficient;
        result.exponent = this.exponent - monomial.exponent;
    } else {
        System.out.println("Divide by 0");
    }
    return result;
}
```

The divide method divides two given monomials. This method takes into account division by 0, printing a message to console, if such a situation happens. This method is used in the main polynomial division algorithm, when the leading terms of both the dividend and the divisor are divided in order to obtain the quotient.

Method – derive

```
void derivate() {
    if (exponent > 0) {
        coefficient *= exponent;
        exponent--;
    } else {
        coefficient = 0;
        exponent = 0;
    }
}
```

The derive method simply derives the current monomial by multiplying the coefficient to the exponent, then subtracting by one the monomial's power. The method takes into account the situation when the exponent is 0.

Method – integrate

```
void integrate() {
    exponent++;
    coefficient /= exponent;
}
```

The integrate method performs the integration of the current monomial. The process is reverse to that of derive: first the exponent is increased, then the coefficient is divided by the exponent in order to obtain the integral.

Polynomial Class

The polynomial class is the main class of the program, essential for performing any mathematical operation on the user-given polynomials. It consists of the members: terms and degree. The terms are stored in an array list. The index of the element in the list matches the exponent of the monomial term that is stored at that index. When there are no terms of a particular power in the given polynomial, but there exist terms of a higher exponent, monomials of coefficient 0 are added up to the next existing exponent. This method allows the index to correspond with the power of the coefficient.

Method – search

```
protected Monomial search(int degree) {
    for (Monomial monomial :
            terms) {
        if (monomial.exponent == degree) {
            return monomial;
        }
    }
    return new Monomial();
}
```

The search method returns the monomial term of a specified degree from the terms list. If none is found, an empty monomial is returned.

Method – addMonomial

```
protected void addMonomial(Monomial monomial) {
    if (this.terms.isEmpty()) {
        for (int i = 0; i <= monomial.exponent; i++) {
            this.terms.add(new Monomial(i, coefficient: 0));
        }
        this.degree = monomial.exponent;
    } else if (monomial.exponent > this.degree) {
        for (int i = this.degree + 1; i <= monomial.exponent; i++) {
            this.terms.add(new Monomial(i, coefficient: 0));
        }
        this.degree = monomial.exponent;
    }
    this.terms.set(monomial.exponent, this.terms.get(monomial.exponent).add(monomial));
}
```

The method adds a monomial term to an already existing polynomial. The method first checks if the monomial exists, then if none is found, it expands the list up to the exponent of the monomial to be added. Lastly, the monomial gets added and the degree of the polynomial gets updated.

Method – add

```
public Polynomial add(final Polynomial polynomial) {
    Polynomial result = new Polynomial();
    for (int i = 0; i <= max(this.degree, polynomial.degree); i++) {
        result.addMonomial(this.search(i).add(polynomial.search(i)));//add terms of degree i to result
    }
    return result;
}
```

The addition of two polynomials is carried out by adding the two term by term, based on the corresponding exponent of the terms to be added. The method also updates the degree of the polynomial and eliminates unnecessary zero monomials in case the result of the addition lowers the degree of the polynomial (i.e. (x+1)+(-x+1)=2)

Method – subtract

```
public Polynomial subtract(final Polynomial polynomial) {
    Polynomial result = new Polynomial();
    for (int i = 0; i <= max(this.degree, polynomial.degree); i++) {
        result.addMonomial(this.search(i).subtract(polynomial.search(i)));
    }
    result.updateDegree();
    return result;
}
```

The subtraction of two polynomials is very similar to the addition. The same term-by-term process is employed, after which the degree is updated and the list rearranged.

Method – multiply

```
public Polynomial multiply(final Polynomial polynomial) {
    Polynomial result = new Polynomial();
    for (Monomial term :
            this.terms) {
        for (Monomial term2 :
                polynomial.terms) {
            result.addMonomial(term.multiply(term2));
        }
    }
    return result;
}
```

Multiplication between two polynomials needs to be broken down into the multiplication of every term of the first polynomial with the multiplictaion of every term of the second polynomial.

Method – divide

```
public ArrayList<Polynomial> divide(final Polynomial that) {
    ArrayList<Polynomial> results = new ArrayList<>();
    Polynomial dividend;
    Polynomial divisor;
    if (this.degree >= that.degree) {...} else {...}
    if (divisor.degree == 0 && divisor.terms.get(0).getCoefficient() == 0) {
        return null;
    } else {
        Polynomial quotient = new Polynomial();
        Polynomial remainder = new Polynomial(dividend);
        while (remainder.degree >= 0 && remainder.degree >= divisor.degree) {
            Monomial term = remainder.getLeadingTerm().divide(divisor.getLeadingTerm());
            quotient.addMonomial(term);
            remainder = remainder.subtract(divisor.multiply(term));
        }
        results.add(quotient);
        results.add(remainder);
    }
    return results;
}
```

Division is performed after the long division algorithm. First we identify the dividend and the divisor, based on which term has a larger degree, since division with a larger polynomial would be pointless, as the quotient would always be 0 and the remainder would be equal to the dividend. After this step, the divisor is asserted to be different than 0, then the quotient and remainder are initialized. While the remainder is more than 0, and bigger than the divisor we continue to divide the leading terms of the two polynomials, and adding the result to the quotient. Then the result multiplied with the divisor is subtracted from the remainder. The result is passed as a list, since we have both a remainder and a quotient.

Method – derive

```
public Polynomial derivate() {
    Polynomial result = new Polynomial( other: this);
    for (Monomial monomial :
            result.terms) {
        monomial.derivate();
    }
    return result;
}
```

Every term of the polynomial is taken and derived, then the resulting term gets updated such that the index corresponds to the term's exponent.

Method – integrate

```
public Polynomial integrate() {
    Polynomial result = new Polynomial( other: this);
    for (Monomial monomial :
            result.terms) {
        monomial.integrate();
    }
    return result;
}
```

The method is similar to the derivative method.

FXMLController Class

The FXMLController is responsible for communicating with the GUI and the model. It contains the methods used to parse user input and display the results of the computations on screen.

Method – getInput():

```
@FXML
private void getInput() {
    PolynomialString polynomialString = new PolynomialString();
    if (currentPolySymbol.contains("P")) {
        polynomialP = polynomialString.parsePolynomial(inputBar.getText());
    } else {
        polynomialQ = polynomialString.parsePolynomial(inputBar.getText());
    }
}
```

The getInput() method is used to parse user input from the text box into usable information. The inputted text is parsed, then stored into either the Polynomial object P or Q, depending on the currentPolySymbol variable.

Method – add/subtract/multiply/derivate/integrate:

```
@FXML
private void add() {
    System.out.println("+");
    currentOp = "+";
    polynomialRes = polynomialP.add(polynomialQ);
    displayCurrentOp();
    selectResult();
}
```

The add/subtract/multiply/derivate/integrate methods follow a very similar structure: change the currentOp variable, make the calculation, display the currentOp variable together with the result so that the user may see it right away. Derivate and integrate are single polynomial operations and they will be performed only on the currently selected polynomial. The other operations will be performed in the order: P <op> Q.

Method – divide:

```
@FXML
private void divide() {
    System.out.println("/");
    currentOp = "/";
    ArrayList<Polynomial> result = polynomialP.divide(polynomialQ);
    if (result == null) {
        remainderBar.setText("CANNOT DIVIDE BY 0!");
    } else {
        displayPolynomial(result.get(0));
        displayPolynomial(result.get(1),  i: 0);
    }
    displayCurrentOp();
}
```

The divide method must also account for the case of dividing with zero, and must find a way to display the error to the user. This is done based on the result variable which contains two Polynomials, the first being the quotient with the second one being the remainder of the division. If the result is null, the division by zero error is displayed. Otherwise, the results are printed into the two result boxes present between the text input and the button pad. The first result box displays the quotient, and is used also to display the results of the other mathematical operations, while the second result box is used only in the case of division.

Method – selectP/selectQ/selectRes:

```
@FXML
private void selectP() {
    System.out.println("P");
    currentPoly = polynomialP;
    displayPolynomial(currentPoly);
    currentPolySymbol = "P";
    displayCurrentPolySymbol();
}
```

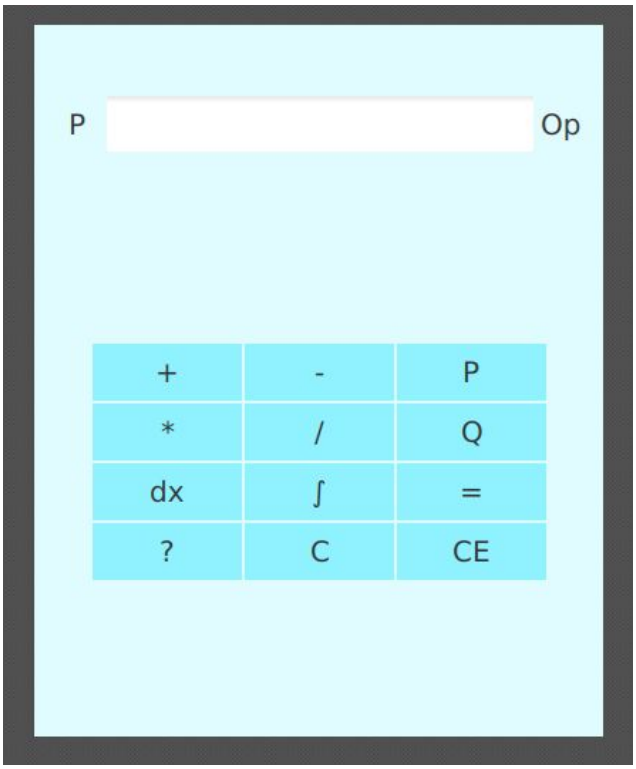
These 3 methods are used when the user wishes to display one of the three polynomials: P, Q or the result. They work based on the same principle: select the current polynomial, then display the current polynomial and its symbol. This allows us to use the same input box for both polynomials P and Q, saving space and making the GUI better looking.

Method – prepareText:

```
private StringBuilder prepareText(Polynomial p) {
    StringBuilder string = new StringBuilder();
    for (int i = p.getTerms().size() - 1; i >= 0; i--) {
        Monomial currentMonomial = p.getTerms().get(i);
        double coefficient = currentMonomial.getCoefficient();
        int exponent = currentMonomial.getExponent();
        if (i != (p.getTerms().size() - 1) && coefficient > 0) {...}
        if (coefficient != 0) {
            if (coefficient == 1 && exponent > 1) {
                string.append(String.format("x^%d", exponent));
            } else if (coefficient == -1 && exponent > 1) {
                string.append(String.format("-x^%d", exponent));
            } else if (exponent == 1) {
                if (coefficient == 1) {
                    string.append("x");
                } else if (coefficient == -1) {
                    string.append("-x");
                } else {
                    if (coefficient % 1 == 0) {
                        string.append(String.format("%.0fx", coefficient));
                    } else {
                        string.append(String.format("%.2fx", coefficient));
                    }
                }
            } else if (exponent == 0) {
                if (coefficient % 1 == 0) {
                    string.append(String.format("%.0f", coefficient));
                } else {
                    string.append(String.format("%.2f", coefficient));
                }
            } else {
                if (coefficient % 1 == 0) {
                    string.append(String.format("%.0fx^%d", coefficient, exponent));
                } else {
                    string.append(String.format("%.2fx^%d", coefficient, exponent));
                }
            }
        }
    }
    return string;
}
```

The method helps print a pretty version of the polynomial stored into the program. It needs to take into account the different combinations between the coefficient and exponent in order to display only necessary information, such as the coefficient (only if different from 1), the exponent (if different from 1), and the symbol x (if the exponent is different from 0). The symbols should also only be displayed if the coefficient is not zero, otherwise there is no reason to pollute the printed output with unnecessary data.

GUI



The GUI was created with the help of Scenebuilder. It implements a text input field, a current polynomial indicator, a current sign symbol, and buttons for every mathematical operation. In order to use the application, the user needs to enter a polynomial in the text box, then select the desired polynomial to store the string into (either P or Q), after which the user can press the enter key on the keyboard and the string will be parsed and saved. Pressing either the P or Q button after inserting the desired polynomials will display the current polynomials stored in P and Q. Pressing the = key will display the current result. The user is able to delete the current polynomial and text input by pressing the C key. Pressing CE will clear everything, basically restoring the calculator to an initial state. The program expects the users to fill in the polynomials first, only after that, will pressing the mathematical operations buttons yield relevant results. The result is calculated instantly when clicking on the mathematical operations. Pressing the = button will then display the result to the user.

5. Results

Using Junit the main operations of the calculator have been tested: addition, subtraction, multiplication, division, derivation and integration. 5 differrent tests have been implemented for each operation, taking into consideration special cases as well.

```
@Test
void test_add1() {
    Polynomial expected = new PolynomialString().parsePolynomial( string: "x^2+x+1");
    Polynomial p = new PolynomialString().parsePolynomial( string: "x^2+1");
    Polynomial q = new PolynomialString().parsePolynomial( string: "x");
    Polynomial result = p.add(q);
    assertEquals(expected, result);
}
```

```
@Test
void test_subtract1() {
    Polynomial expected = new PolynomialString().parsePolynomial( string: "5x^2+4x+1");
    Polynomial p = new PolynomialString().parsePolynomial( string: "3x^3+5x^2+6x+2");
    Polynomial q = new PolynomialString().parsePolynomial( string: "3x^3+2x+1");
    Polynomial result = p.subtract(q);
    assertEquals(expected, result);
}
```

```
@Test
void test_multiply1() {
    Polynomial expected = new PolynomialString().parsePolynomial( string: "x^2-x-2");
    Polynomial p = new PolynomialString().parsePolynomial( string: "x+1");
    Polynomial q = new PolynomialString().parsePolynomial( string: "x-2");
    Polynomial result = p.multiply(q);
    assertEquals(expected, result);
}
```

```
@Test
void test_divide1() {
    Polynomial expectedQuotient = new PolynomialString().parsePolynomial( string: "x");
    Polynomial expectedRemainder = new PolynomialString().parsePolynomial( string: "2");
    Polynomial p = new PolynomialString().parsePolynomial( string: "x^2+2");
    Polynomial q = new PolynomialString().parsePolynomial( string: "x");
    ArrayList<Polynomial> result = p.divide(q);
    assertEquals(expectedQuotient, result.get(0));
    assertEquals(expectedRemainder, result.get(1));
}
```

6. Conclusions

The calculator may be improved in the following ways:

- Refactoring certain elements of code, while deobfuscating others
- Implementing a help page via the button left unused in the UI
- Feedback to the user, when inputting an incorrect string
- More testing via Junit
- More intuitive user experience by allowing the user to calculate the results of the polynomials only after pressing the = key.

During the creation of this project I have learned how to better structure my time so as to meet deadlines, furthermore testing with Junit has taught me a much faster and easier way to check if the methods implemented by me work. Working for the first time with gradle also reduced the initial set-up time. This made for a very convenient installation of JavaFx, and Junit without needing to link the library paths manually.

7. Bibliography

https://regex101.com/
https://www.tutorialspoint.com/javafx/javafx_layout_panes.htm
https://www.jetbrains.com/help/idea/components-of-the-gui-designer.html
https://rosettacode.org/wiki/Polynomial_long_division
https://docs.oracle.com/javafx/2/
https://stackoverflow.com/questions/
https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html
https://www.tutorialspoint.com/javafx/javafx_css.htm
https://docs.oracle.com/javase/8/scene-builder-2/get-started-tutorial/overview.htm#JSBGS164
https://docs.oracle.com/javase/8/scene-builder-2/get-started-tutorial/jfxsb-get_started.htm#JSBGS101
https://docs.oracle.com/javase/8/scene-builder-2/user-guide/index.html
https://docs.oracle.com/javase/8/scene-builder-2/work-with-java-ides/index.html