MINISTERUL EDUCAȚIEI NAȚIONALE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Polynomial Calculator

Fundamental Programming Techniques

Horvath Andrea Anett

Faculty of Automation and Computer Science
Computer Science Department
2'nd year, Group 30424

Contents

I. Objectives of the proposed laboratory work

The main objective of the proposed laboratory work was to develop a Java application that performs the following mathematical operations on polynomials:

- addition
- subtraction
- multiplication
- division
- differentiation
- integration

As secondary objectives, the following can be considered:

- developing the application in accordance with the object oriented programming principles
- getting familiar with the MVC (Model - View - Controller) structure
- choosing the right data structures and algorithms for the desired approach
- drawing the UML diagrams and writing the code based on them (forward engineering)
- determining all the use-cases and assuring that the program takes into account all the possibilities
- developing a graphic user interface (GUI) using Java Swing
- verifying the validity of the input with Java Regular Expressions
- unit testing with JUnit

II. Analysis of the laboratory work, modelling, scenarios and use-cases

Analysis

The polynomial calculator will be designed and implemented with a dedicated graphic user interface through which the user can enter polynomials, choose the operation to be performed (addition, subtraction, multiplication, division, differentiation, integration) and display the results.

The application is required to operate with polynomials of one variable and integer coefficients. These constraints are defining for the development of the application, since the input must be validated. Some operations do not output a polynomial with integer coefficients, so the model must be developed in such a way that the result can have real coefficients.

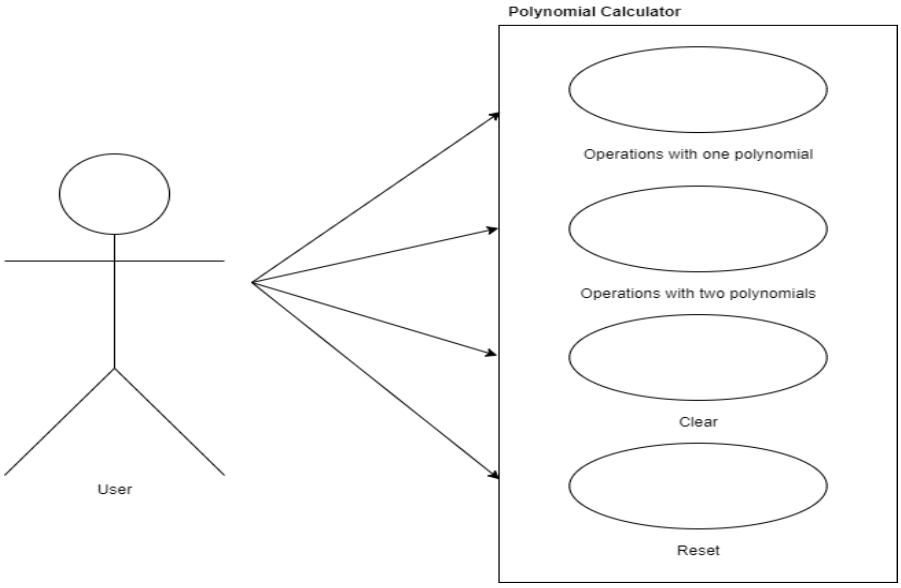
In the end, the final application must be tested. Regarding the type of testing to be used, the unit testing is chosen. This will be done using the Java framework JUnit.

Modelling

The polynomial calculator will be modeled as a MVC (Model - View - Controller) structure application. By this, it is desired to separate the application into a Model, which contains the operations on polynomials and the implementation of the polynomials, a View, which creates the display, the graphic user interface, and a Controller, which responds to user requests, interacting with both the View and Model as necessary.

Scenarios and use-cases

The possible use-cases are summarized in the next diagram and detailed below.



In all the following use-cases the primary actor is considered to be the user.

1'st use-case: *Operations with one polynomial*

Note: in this use-case the input of the second polynomial is ignored and the text field will be set to null.

Main success scenario:

1. The user introduces the polynomial in the first polynomial text field.
2. The user clicks on one of the buttons matching one of the operations with one polynomial (Differentiate/Integrate).
3. The application checks the validity of the polynomial.
4. The result is shown in the result text field.

Alternative sequence A:

1. The user introduces an invalid polynomial in the first polynomial text field.
2. The user clicks on one of the buttons matching one of the operations with one polynomial (Differentiate/Integrate).
3. The validation of the polynomial fails.
4. The application displays an error message, i. e., "First polynomial is invalid!".
5. The flow resumes from step 1.

Alternative sequence B:

1. The user lefts the first polynomial text field empty.
2. The user clicks on one of the buttons matching one of the operations with one polynomial (Differentiate/Integrate).
3. The application automatically fills the first polynomial text field with 1.
4. The flow continues from step 3.

2'nd use-case: *Operations with two polynomials*

Main success scenario:

1. The user introduces the polynomials in the first and second polynomial text field.
2. The user clicks on one of the buttons matching one of the operations with two polynomials (Add/Subtract/Multiply/Divide).
3. The application checks the validity of the polynomials.
4. The result is shown in the result text field (in the remainder text field as well for division) .

Alternative sequence A:

1. The user introduces the polynomials in the first and second polynomial text field and one or both are invalid.
2. The user clicks on one of the buttons matching one of the operations with two polynomials (Add/Subtract/Multiply/Divide).
3. The validation of the polynomial fails.
4. The application displays an error message, i. e., "First polynomial is invalid!", " Second polynomial is invalid!", "The entered polynomials are invalid!".

5. The flows resumes from step 1.

Alternative sequence B:

1. The user lefts the first or the second polynomial text field empty.
2. The user clicks on one of the buttons matching one of the operations with two polynomials (Add/Subtract/Multiply/Divide).
3. The application automatically fills the empty polynomial text field with 1.
4. The flow continues from step 3.

Alternative sequence C:

1. The user introduces "0" in the second polynomial text field.
2. The user clicks on the button Divide.
3. The application displays an error message, i. e., "Division by 0 (zero) cannot be performed!".
4. The flow resumes from step 1.

3'rd use-case: *Clear*

Main success scenario:

1. The user clicks on the button Clear.
2. The application sets all the text fields to null.

No alternative sequence.

4'th use-case: *Reset*

Main success scenario:

1. The user clicks on the button Reset.
2. The application sets the polynomial text fields to their original value and the result and remainder text fields to null.

No alternative sequence.

III. Development

1. Development approach

The polynomial calculator application is developed, as before mentioned, in a MVC (Model - View - Controller) manner. Thus, the project includes three packages: Model, View and Controller.

The package Model includes the classes:

- Fraction: contains the mathematical operations using fractions;
- Monomial: contains the representation of a monomial as a group formed by an integer representing the degree and a fraction representing the coefficient;
- Polynomial: contains the representation of a polynomial as an array list of monomials and the required mathematical operations performed on this type.

The package View includes only the class View which contains the necessary Swing components, the constructor which deals with the visual placement of the objects on the panel, methods for resetting and clear the text fields, a method to display errors and the methods for adding the listeners for the buttons.

The package Controller includes the classes:

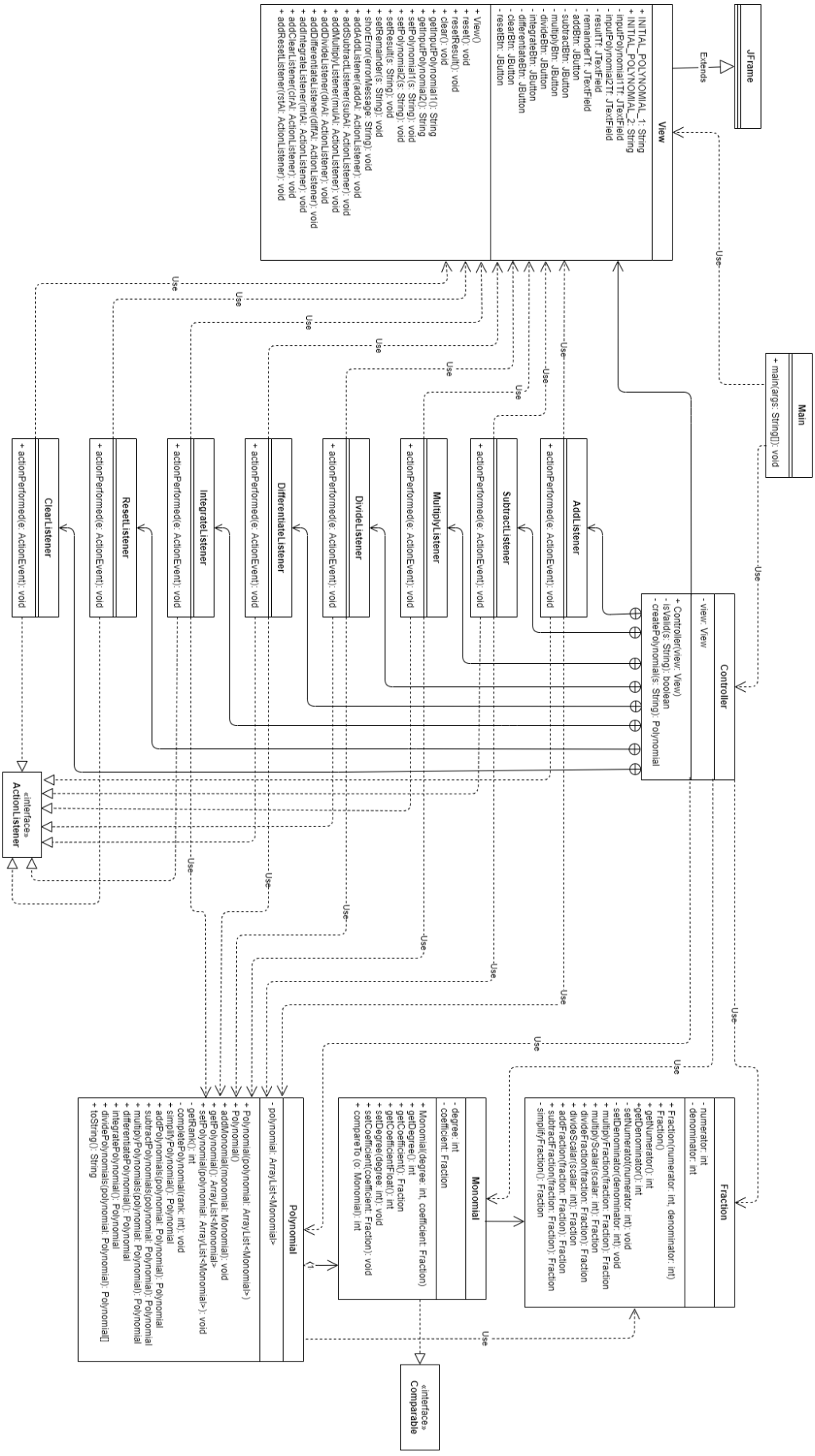
- Controller: contains the constructor which sets a view visible and adds the listeners, inner classes which implement the listeners and also methods for validating and creating a polynomial out of the input string;
- Main: contains only the main method which calls the Controller to do its job

2. Structure and UML Diagrams

2.a. Package Diagram

MVC model; Controller in charge of the Model and the View. The content of the packages is shown in the class diagram from 2.b.

2.b. Class Diagram



IV. Implementation

1. Model package

1.a. Class Fraction

Instance variables: *numerator*: int, *denominator*: int.

In its methods the mathematical operations on fractions are implemented, in order to use them in class Polynomial, to be able to work with the coefficients of type Fraction.

Contains constructors, getters and setters in order to have access to the private instance variables: numerator, denominator.

An essential operation to be performed on fractions is the **simplification**.

```
private Fraction simplifyFraction() {...}
```

This method uses the algorithm for finding the greatest common divisor for the numerator and the denominator and if it finds one, it divides each of them with this value.

The following method implements the **addition** of two fractions. It takes as argument the second fraction, and computes the result after the formula a/b + c/d = (a*d + c*d)/b*d. The result is returned after simplification.

```
public Fraction addFraction(Fraction fraction) {
                Fraction result = new Fraction();
                result.numerator = (this.numerator * fraction.denominator) + (fraction.numerator *
this.denominator);
                result.denominator = this.denominator * fraction.denominator;
                return result.simplifyFraction();
        }
```

The result of the following methods is computed similarly with the one presented above: **multiplication**, **multiplication by scalar**, **division**, **division by scalar**, **subtraction**.

1.b. Class Monomial

Instance variables: *degree*: int, *coefficient*: Fraction.

These variables represent the attributes of a monomial: coefficient*X^degree (ex. 2X^3).

Contains constructors, getters and setters in order to have access to the private instance variables: degree and coefficient. Besides the standard getters, it also contains the next one:

```
public int getCoefficientFloat() {...}
```

This is used to get the value of the coefficient as a real number.

The class implements the Comparable interface, and contains the next method which has the purpose of ordering the polynomial in ascending order of monomial degrees.

```
@Override
	public int compareTo(Monomial o) {
		return this.degree - o.getDegree();
	}
```

1.c. Class Polynomial

Instance variables: *polynomial*: an ArrayList of Monomials.

Contains constructors, getters and setters in order to have access to the polynomial.

The method `public void addMonomial(Monomial monomial) {...}` is used to insert a given monomial in the polynomial. Uses the methods available in java.util.ArrayList.

The method `private int getRank() {...}` returns the rank of the polynomial by finding the monomial in the array list with the maximum degree.

The method `private void completePolynomial(int rank) {...}` is used when the polynomial we have does not contains all the terms, monomials with certain degrees smaller than a given rank are missing. The method searches monomials in the polynomial with the degrees in the range [0, rank] , and if does not finds any, it inserts a null coefficient monomial of that degree. In the end, the polynomial is sorted using java.util.Collections.

The method `public Polynomial simplifyPolynomial() {...}` is used to reduce the polynomial. It adds the monomials with the same degree and eliminates the zero coefficient monomials. For adding the equal degree terms it constructs a new polynomial of the same rank as the initial polynomial with zero coefficients, and adds all the monomials from the initial polynomial with the corresponding monomial in range [0, rank] in the new polynomial. Then, the terms with non zero coefficients from this new polynomial are copied into a final polynomial which is returned.

The method `public Polynomial addPolynomials(Polynomial polynomial) {...}` performs the addition of two polynomials. The method creates a result polynomial of rank equal with the maximum of the two polynomials to be added. Considering that the received polynomials have no more than one monomial of one degree, we call completePolynomial() method. Then, being complete, ordered and having no duplicate terms, the result is obtained by adding the terms in the two polynomials termwise. The result is simplified in case some terms will cancel, and then returned.

The method `public Polynomial subtractPolynomials(Polynomial polynomial) {...}` performs the subtraction of two polynomials. The method works in the same manner as the addition, with the exception that instead of adding the terms of the two polynomials, they are subtracted.

The method `public Polynomial multiplyPolynomials(Polynomial polynomial) {...}` performs the product of two polynomials. It constructs a result polynomial, empty at the beginning. In this, the product of each two monomials from the two polynomials is added in a double foreach loop. The result in then simplified and returned.

The method `public Polynomial differentiatePolynomial() {...}` uses only one polynomial. It adds in a new created result polynomial monomials obtained from the differentiation of each monomial from the given polynomial. For computing the coefficient and the degree of the monomials to be inserted, the basic rule is used: (c*X^d)' = c*dX^(d-1). The terms with degree 0 in the initial polynomials are ignored because those are constants and the derivative of them equals to 0, and also because the algorithm will return a monomial with degree -1 instead of a null term. In the end, the simplified result is returned.

The method `public Polynomial integratePolynomial() {...}` is similar to the previous one with the following exceptions. Each monomial is integrated after the formula: $\int(c*X^d)dX = c/(d+1)*X(d+1)$. No term is ignored.

The method `public Polynomial[] dividePolynomials(Polynomial polynomial) {...}` is the most complex one involving a more complex algorithm which returns two polynomials representing the quotient and the remainder. Two polynomials are created for the quotient and the remainder. The quotient will initially be empty, and the terms will be added at each iteration of the while loop. The remainder gets the value of the dividend. At each iteration of the loop, the leading term (the monomial with the higher degree) of the reminder and the divisor is determined. In the quotient, the term to be added is the result of the division of the two leading monomials, and from the remainder is subtracted this result multiplied by the divisor. The while loop exits when the remainder has the rank smaller than the divisor or all of its terms are reduced and becomes empty. In the end, the quotient and the remainder are copied in the array of polynomials to be returned.

The last method of this class is `public String toString() {...}` which converts a polynomial into a string in order to be later displayed. The method is written to return the simplest alternative of writing a polynomial. If the coefficient (numerator/denominator) is 1 or -1 than the method only displays the sign. If the denominator of the coefficient is 1, the method only displays the numerator. Does not return a string starting with a "+" if the polynomial starts with a positive term. Does not displays the "^1" after "X" if the degree of the monomial is 1 or if the degree is 0 than the "X^0" is omitted. If the polynomial is empty than the method displays a "0".

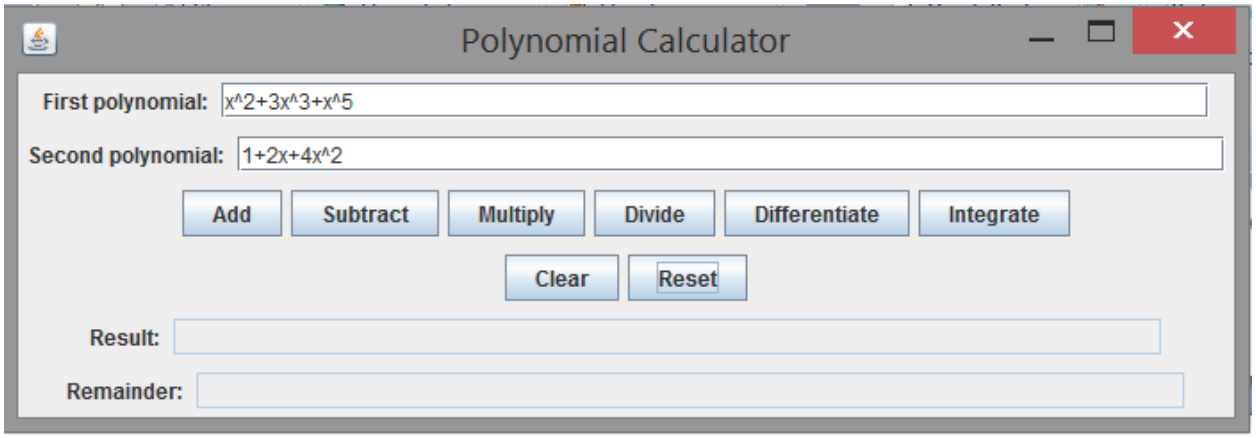
1. View package

Class View

Instance variables: 4 JTextFields for the two polynomials, result and remainder; 8 JButtons: 6 for the mathematical operations (Add, Subtract, Multiply, Divide, Integrate, Differentiate) and 2 for the Clear and Reset operations.

The method contains two constant strings to be set at the beginning in the polynomial text fields.

The class contains a constructor which instantiates 6 JPanels on which the text fields and the buttons are placed. There also is a final panel which incorporates all the previous ones.



Contains getters and setters in order to have access to the text fields.

The method `public void reset() {...}` resets the interface. Sets the text fields of the result and remainder to null and the polynomials to the constant values.

The method `public void resetResult() {...}` only sets the text fields of the result and the remainder to null.

The method `public void clear() {...}` sets all text fields to null.

There is also the following method for displaying errors:

```
public void showError(String errorMessage) {
                JOptionPane.showMessageDialog(this, errorMessage);
        }
```

And in the end, there are the methods for adding listeners for each button corresponding to one operation. All of them are similar. The one for the Add button is shown below:

```
public void addAddListener(ActionListener addAl) {
                addBtn.addActionListener(addAl);
        }
```

3. Controller package

3.a. Class Controller

Instance variables: *view*: View.

The constructor of this class sets the view visible and adds the listeners for the buttons in the view it generates.

The method `private boolean isValid(String s) {...}` is receiving a string and it returns an answer to the question whether the given string represents a polynomial which can be converted into an object of type Polynomial. The method uses a regular expression to find monomials in the given string and concatenates the results in an auxiliary variable of type String. If the initial string is identical with the auxiliary one, than the string represents an ArrayList of monomials and can be converted into a polynomial. The method validates only polynomials with integer coefficients.

The method `public Polynomial createPolynomial(String s) {...}` will be called only after the previous one was called. This uses the same regular expression as the method above in order to find the monomials in the string, and also uses a second expression to parse each monomial found with the first expression into coefficient and degree, values to be included in the polynomial. The coefficients will be added in the polynomial as a fraction with the denominator equal to 1. The obtained polynomial is simplified and returned.

The class contains 8 inner classes which implement the interface ActionListener. Each of them contains a method which performs one of the operations required when one of the buttons is clicked.

In `class AddListener implements ActionListener {...}` there is a method `public void actionPerformed(ActionEvent e) {...}` in which the two strings from the polynomial text fields are validated and then the two polynomials are created. In case of invalid polynomials, an error message will be shown. The sum of the two polynomials is computed, converted to a string and displayed in the result text field. All of this is done in a try-catch block.

The other 7 inner classes are implemented similarly and will not be presented. The only think that worth to be mentioned is that the method in the DivideListener class checks for the second polynomial to be non zero.

3.b. Class Main

Contains only the main method and calls the constructor of the controller on a view.

V. Results

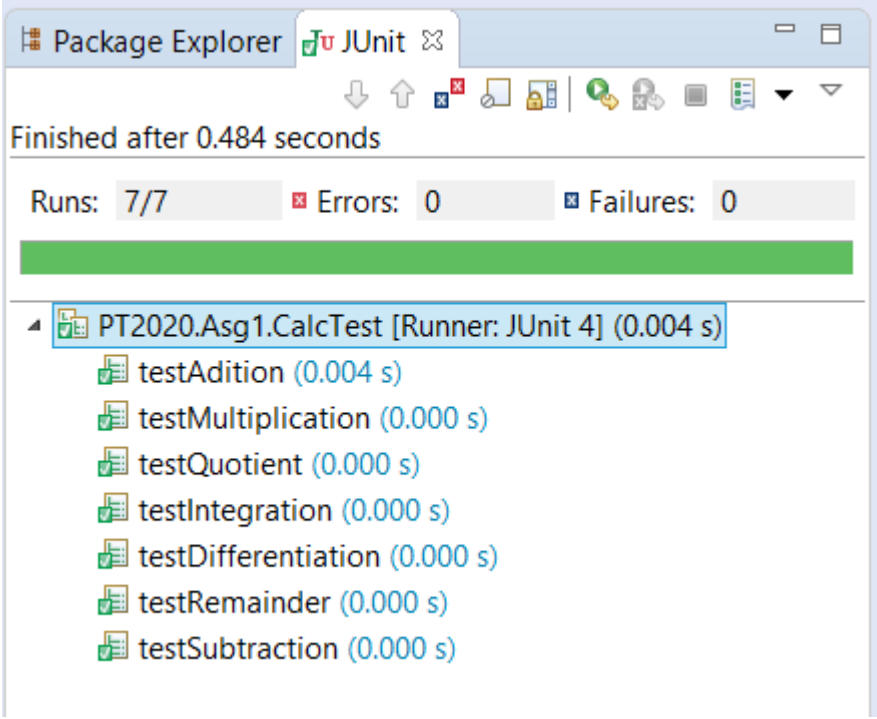
For testing the correctness of the mathematical operations performed on polynomials and also the validation and the conversion of the string to a polynomial, JUnit was used.

A test class named CalcTest was implemented and contains 7 methods for testing all the possible results (Addition, Subtraction, Multiplication, Quotient, Remainder, Integration, Differentiation) made on the two initial values of the polynomials given as constants in the View class.

The classes are similar so only the one which tests the addition will be presented:

```
@Test
        public void testAdition()
    {
                Polynomial poly1 = controller.createPolynomial("x^2+3x^3+x^5");
                Polynomial poly2 = controller.createPolynomial("1+2x+4x^2");
                String sum = "1+2X+5X^2+3X^3+X^5";
                assertEquals(sum, poly1.addPolynomials(poly2).toString());
    }
```

All the tests were successful.



V. Conclusions

The Polynomial Calculator application fulfils all the requirements of the laboratory work.

For future improvements the following can be considered:

- Operations with more than two polynomials
- Operations with polynomials with more than one variable
- Graphic representations of the polynomials

VI. Bibliography

1. https://www.leepoint.net/GUI/structure/40mvc.html

2. https://en.wikipedia.org/wiki/Polynomial_long_division

3. https://mkyong.com/tutorials/junit-tutorials/

4. https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html

5. https://docs.oracle.com/javase/tutorial/uiswing/

6. http://coned.utcluj.ro/~salomie/PT_Lic/