

POLYNOMIAL PROCESSING

~ Homework 1 ~

Papita Anda

Group 30411

Programming techniques

CONTENTS

1. The objective of the homework.....	3
2. Problem analysis, modelling, scenarios, use-cases.....	3
3. Design (design decisions, UML diagrams, data structures, class design, interfaces, relations, packages, algorithms, user interface).....	4
4. Implementation.....	5
5. Results.....	5
6. Conclusions.....	6
7. Bibliography.....	7

1. *Objective*

The objective of this laboratory homework is the design and implementation of a polynomial processing system. The polynomials are in one-variable form with integer coefficients.

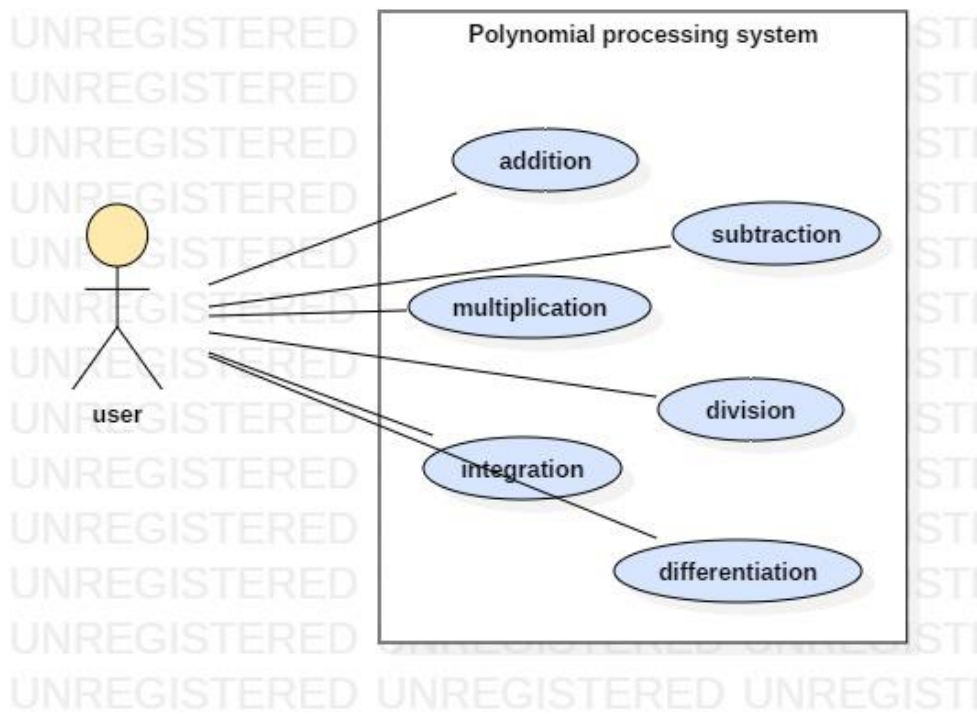
The purpose of this app is effectuating basic operations on polynomials. The user can insert 2 polynomials on which he can choose to perform one or more of the following operations:

- Addition
- Subtraction
- Multiplication
- Division
- Integration
- Differentiation

2. *Problem analysis, modeling, scenarios, use-cases*

The program must correctly effectuate a total number of 6 operations and preserve Objected Oriented paradigms. The complexity of the problem consists in choosing how to implement the polynomial. In this sense, an abstract data type for single-variable polynomials (monomials) was defined. It takes into consideration the coefficient and exponent of the monomial, based on which all operations can be more or less easily designed to perform functionally. A monomial list of elements will compose, finally, the polynomial. This would represent the model part of an MVC pattern. The view part is meant for designing the interface with which the user will be interacting. Two polynomials will be introduced by the user in an appropriate format, on which the operations of addition, subtraction, division and multiplication will be performed. The integration and differentiation will only target one polynomial (the first one in this program). The controller will make the necessary connections between the graphic interface and the model. In addition to these features, the program contains a testing unit implemented with the help of JUnit.

The following diagram represents the choices the user can execute in this app:

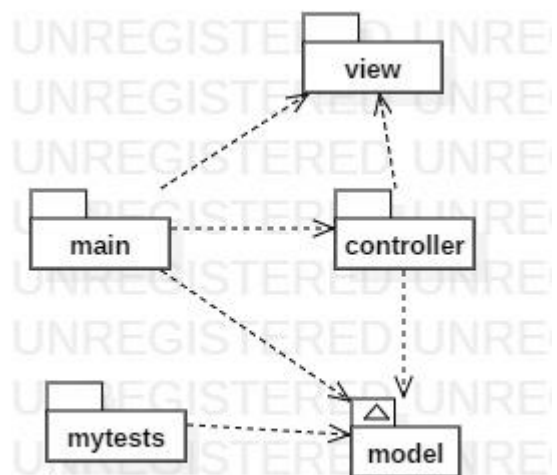


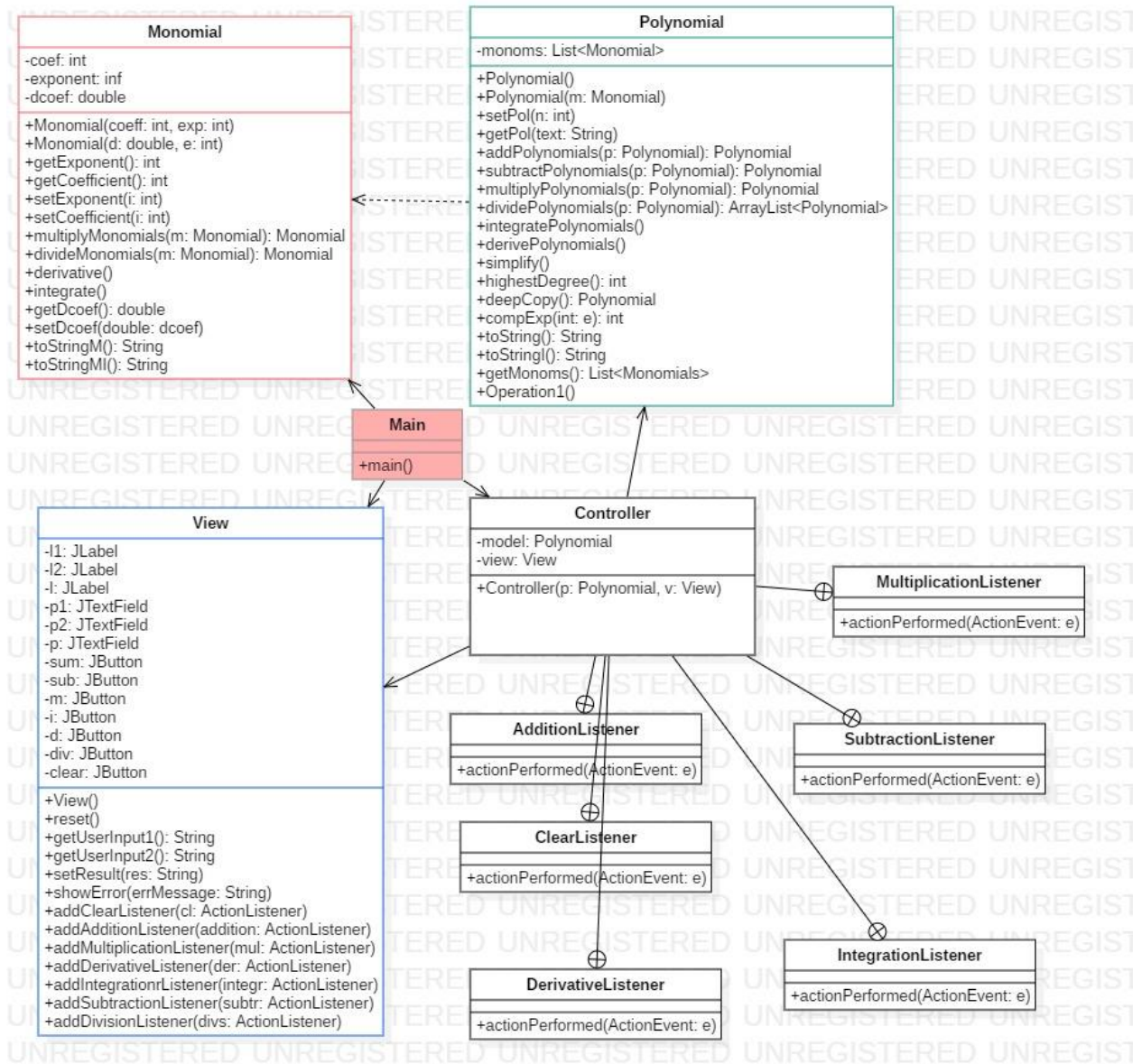
3. Design

The design of this app follows the MVC pattern. Therefore, the model part is represented by the classes Monomial and Polynomial, which are found in the Model package. These classes implement the operations the user wishes to perform on the polynomials of his/her choice. The view part, represented by the class View from the view package is responsible for the graphic user interface design. It extends the JFrame class. Finally, the Controller part of the app acts on both model and view and it keeps them separate. It also controls the data flow into model object (polynomial) and updates the view whenever data changes. More precisely, it coordinates each button the user can press to perform a specific action (operation) and helps display the resulting polynomial into a readable form to the user. In addition to these elements, the app also contains two JUnit testing classes in the tests package, which check for the correct effectuation of the operations inside Polynomial and Monomial classes.

The connections between classes and packages can be seen in the following UML diagrams:

Package relationships





4. Implementation

Monomial class: this class represents the format of the polynomial terms. For example, “ $2x^2$ ” is a monomial (polynomial with a single term). It contains three attributes, `coef`, `exponent` and `dcoef`. The third one, which is a double and not an int like the other two, is used just in the cases of integration and division, when the coefficients might not be integer numbers. Thus, this class has two constructors, one

for monomials with integer coefficients and one for double coefficients. Also, there are three pairs of getters and setters, one for each attribute.

Methods:

1. **public** Monomial multiplyMonomials(Monomial n) -> returns the product between two monomials. Its purpose is to help the multiplication function in the Polynomial class. It simply operates by multiplying the coefficients and then the exponents of each monomial.
2. **public** Monomial divideMonomials(Monomial m) -> returns the result of the division of two monomials by dividing their coefficients and subtracting their exponents. It also serves as a helper function for the Polynomial class.
3. **public void** derivative() -> calculates the derivative of a monomial by subtracting the value '1' from its exponent and multiplying its coefficient by its old exponent. It represents a helper function for polynomials differentiation.
4. **public void** integrate() -> it is very much like the derivative function, it just operates in the opposite way: it divides the coefficients of the monomial and add the value '1' to the exponent. Since the result of the division might not be integer, it saves the value of the new coefficient in the dcoef attribute of the monomial. This function is also meant to help the integration method in the Polynomial class.
5. **public** String toStringM() && **public** String toStringMI() -> these two functions are almost identical. They both turn the monomial term into a valid format (represented by a String) to appear appropriate for the user. The only difference is that one shows the int coefficients, and the other one the double ones (as needed in integration).

Polynomial class: this class represents the polynomial structure. It is composed of a list of monomials called monoms, which contains each term of the polynomial, which together actually make up a polynomial. It has two constructors, one which initializes the ArrayList and another one which can add a monomial to an empty polynomial. It has a getter and a setter for the only attribute in the class, the list of monoms. All the other methods will be presented below.

Methods:

1. **public void** setPol(int[] n) -> this method sets the values of the polynomial's coefficients based on the elements of a vector of ints given as parameter and the exponents are set based on the vector's length, decreasingly. First exponent will have the value n.length, which will decrease by 1 with each exponent. This function helps the tests.

2. `public void getPol(String text)` -> this method extracts the values of the exponents and coefficients from a polynomial given as a String. Firstly, it separates each monomial (term) by splitting the string relative to the symbol '+'. Then, it separates the characters from each term into new Strings, by splitting the numbers (coefficients and exponents) from the other symbols('x' and '^'). The first position will hold a coefficient and the second one an exponent. This monomial will be added to the polynomial and so on with the rest. The method helps extracting the polynomial from the user input.
3. `public Polynomial addPolynomials(Polynomial p)` -> this method perform the addition of this class' instance to a polynomial p. It iterates through the list of the class' polynomial, calls another function to compare the exponents based on their indexes and if they are not the same, it simply adds to the resulting polynomial the term of this polynomial. If they are the same, the coefficients of the two polynomials are added together and the new monomial will be added in the new resulting polynomial. Also, the term will be removed from both polynomial after it is added, in order not to get added twice. At the end, the method iterates through the list of the second polynomial(p) to add what is left in it.
4. `public Polynomial subtractPolynomials(Polynomial p)` -> this method works exactly like the addition, but instead of adding the polynomials' coefficients, it subtracts them. In addition, while iterating through the second polynomial(p), it multiplies its coefficient by '-1', like it would be subtracted from the value '0'. It returns a polynomial with the result.
5. `public Polynomial multiplyPolynomials(Polynomial p)` -> this multiplies two polynomials by iterating through both their monoms lists, then multiplying each monomial. The method calls the monomial multiplication method, described before.
6. `public ArrayList<Polynomial> dividePolynomials(Polynomial p)`
7. `public Polynomial deepCopy()` -> this method is meant to create a "deep" copy of the polynomial, which means not just copying its reference. It is meant to help with the division
8. `public int highestDegree()` -> this method finds the maximum degree that is in a polynomial and returns its value. It is meant to help with the division function
9. `public void derivePolynomial()` ->this method performs the operation of differentiation on a polynomial by differentiating each of his terms(monoms). It calls a method from the Monomial class.
10. `public void integratePolynomial()` -> this method performs the operation of integration and works exactly the same as the previous method
11. `public void simplify()` -> this method is used to reduce the terms of the polynomial when it is possible. Thus, it searches for terms with the same

exponent and adds/subtracts them. It also uses a Collections method, sort, to sort the terms in reverse order by their exponent.

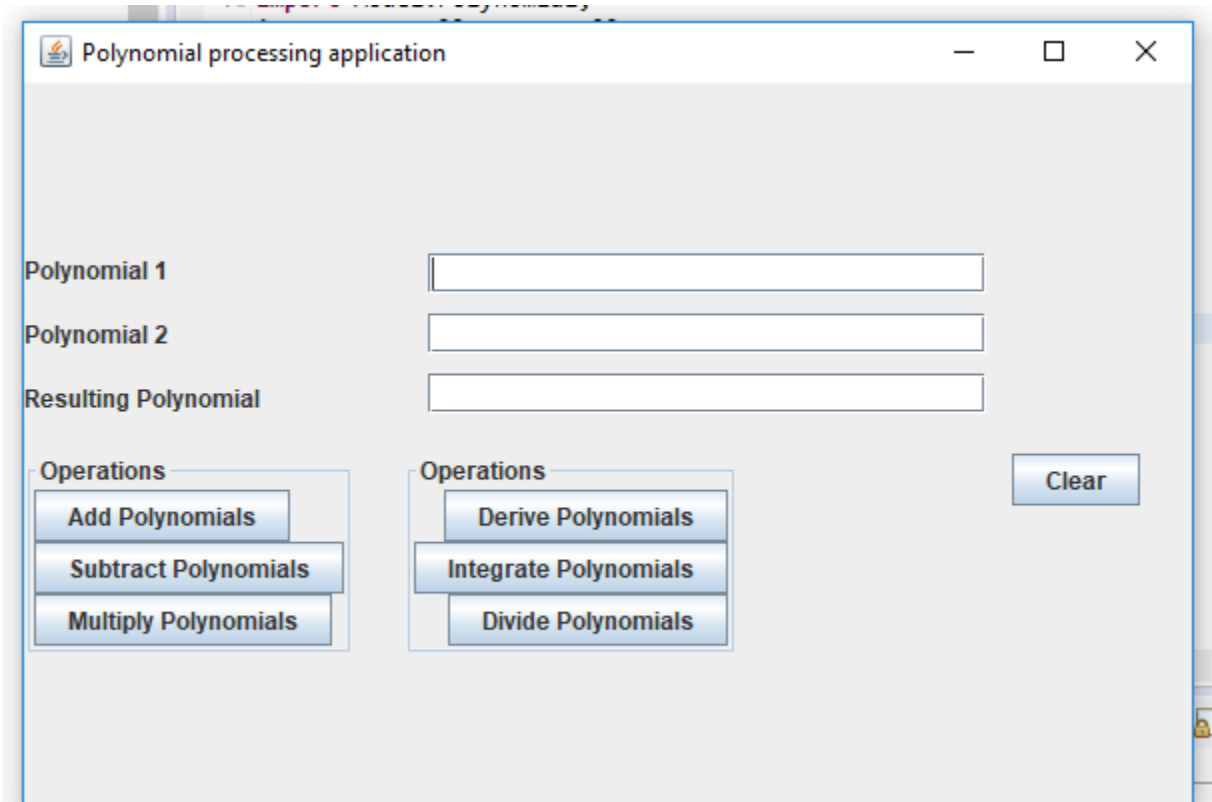
12. `public int compExp(int e)` -> this method compares the exponent of a polynomial with the given parameter e and if they are equal it returns the index of the corresponding monomial. Otherwise, it returns '-1'. It is used in the addition method.
13. `public Monomial maxMonomial()` -> method that finds the monomial with the maximum degree, meant to be used in division
14. `public String toString()` and `public String toStringI()` -> methods that turn the polynomial into a String format that is meant to be displayed on the interface, for the user to see. It calls the toString methods from the Monomial class for each monomial (terms) in the polynomial.

View class: this class creates the interface the user works with regarding to performing polynomial operations. It extends the JFrame class. Its components include labels, text fields and buttons. This time I found the most appropriate and perhaps convenient to use the GridBagLayout with respect to the main panel organization. I set constraints to each component. They refer to the number of columns and lines of the panel. Thus, the panel can be seen as an imaginary matrix and each element is placed in one matrix element. Two more panels were created. They contain the operations buttons and were placed as the other elements in the main panel. Everything is arranged within the constructor, View().

Additional methods:

1. `public void reset()` -> method that turns the text fields empty
2. `public String getUserInput1()` and `public String getUserInput2()` -> these methods get the polynomials introduced by the user
3. `public void setResult(String res)` -> sets the resulted polynomial in its designated area
4. `public void addAdditionListener(ActionListener addition)` -> creates an event for the addition button. All the other buttons are configured in this way and the methods will not be mentioned.

The interface has the following structure:



Controller class: this class acts on both Polynomial and View classes. It controls the data flow that goes into Polynomial and updates the View whenever data changes. More precisely, it coordinates each of the operation buttons to perform a specific task if the user chooses to press them. It consists of one constructor that implements view functions. The class also contains multiple listener classes for each operation. They specify the action a specific button needs to perform.

For example:

```

class SubtractionListener implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        String userInput1 = "";
        String userInput2 = "";
        String res = "";
        Polynomial p1 = new Polynomial();
        Polynomial p2 = new Polynomial();
        try {
            userInput1 = view.getUserInput1();
            userInput2 = view.getUserInput2();
            p1.getPol(userInput1);
            p2.getPol(userInput2);
            model = p1.subtractPolynomials(p2);
            model.simplify();
            res = model.toString();
            view.setResult(res);
        } catch (NumberFormatException nfx) {
            view.showError("Bad input expression in: " + userInput1 + " or " + userInput2);
        }
    }
}

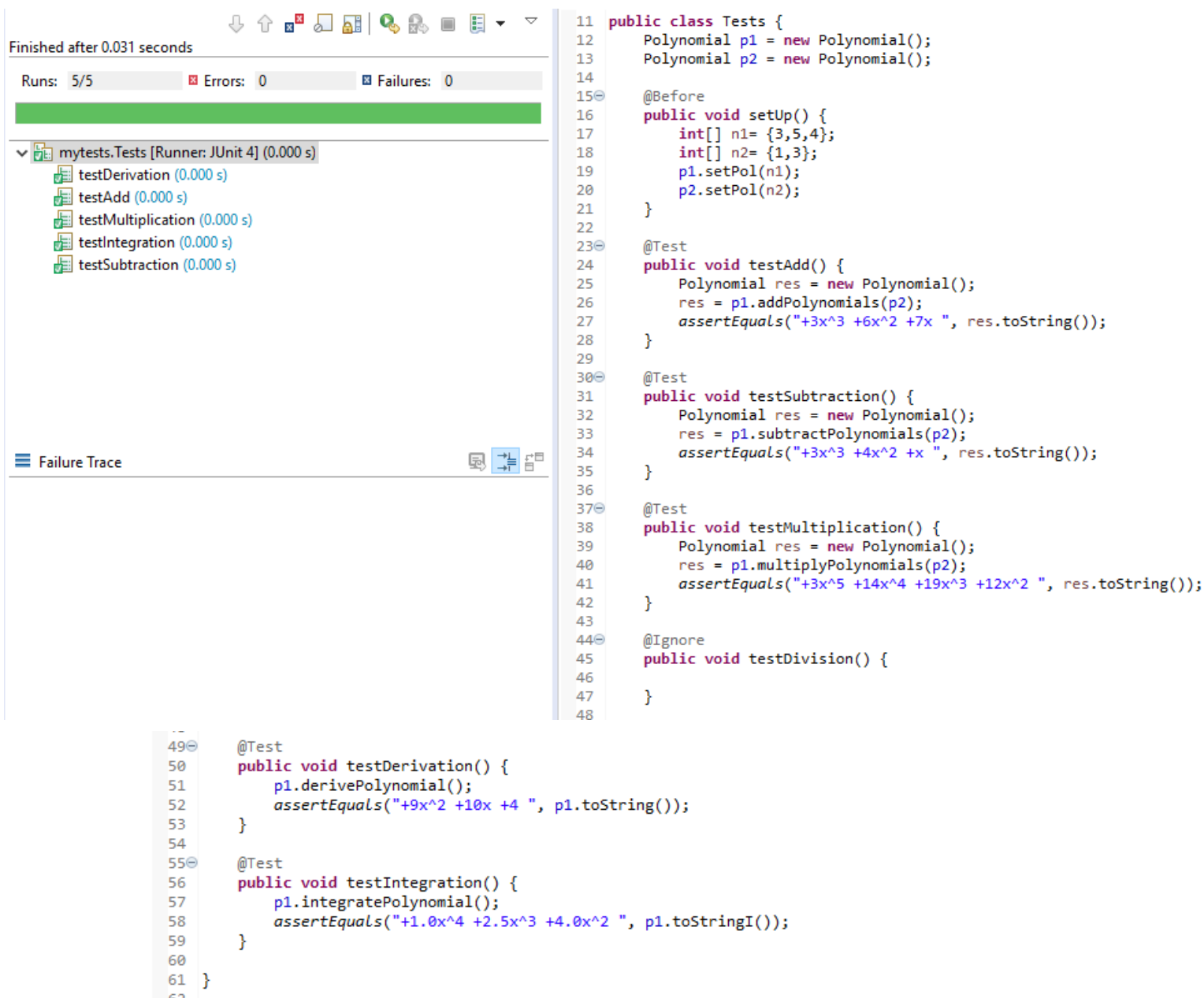
```

Main class: this is the class that starts the application. It only contains a main() function and it creates a model, a view and a controller.

5. Results

Some scenarios were tested in the Tests and TestMon classes, in which both the polynomial and monomial operations are tested.

In the Tests class, operations on polynomials were tested by setting two polynomials with fixed values using the setPol method, as follows:



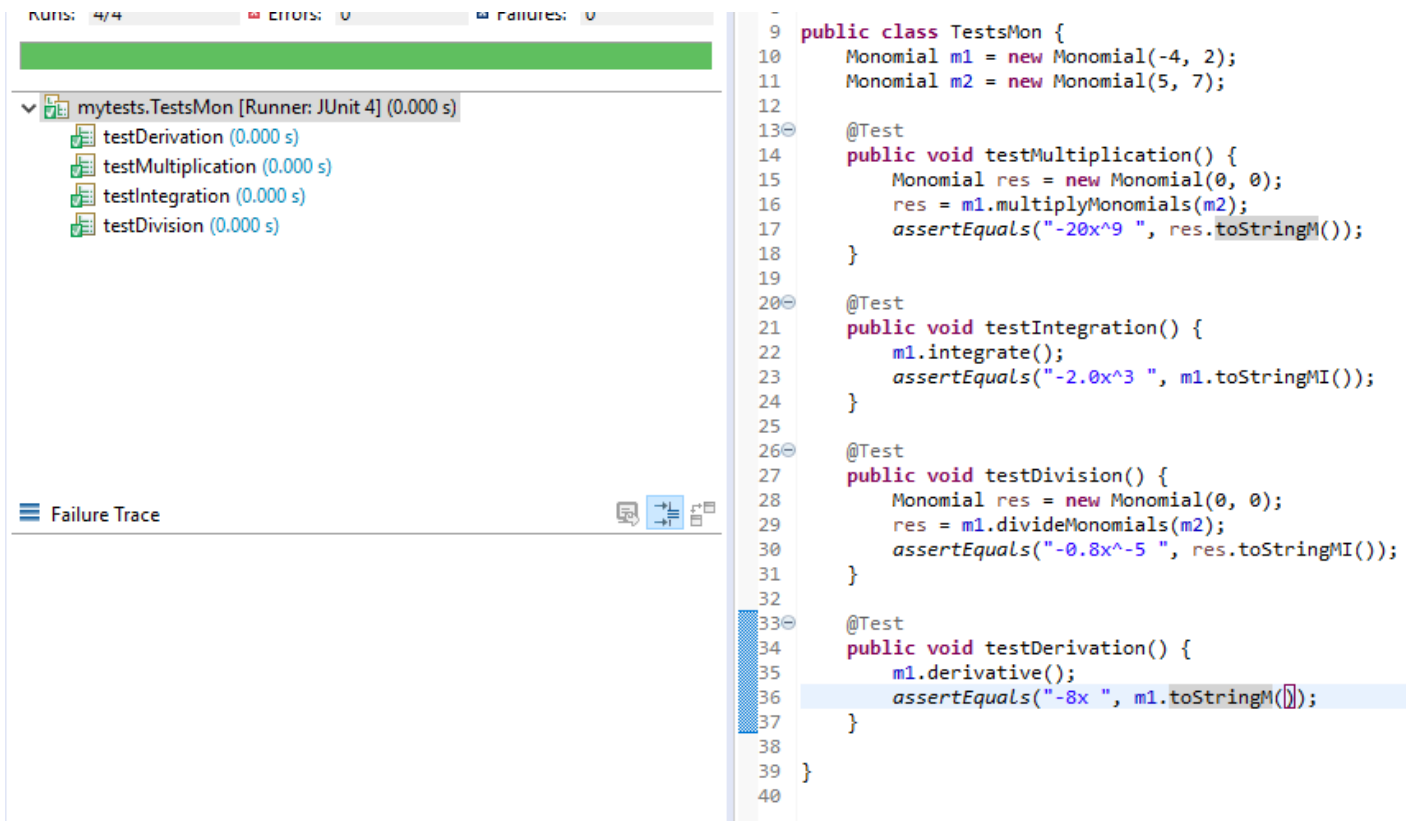
The screenshot displays an IDE interface. On the left, a 'Runs' tab shows test results for 'mytests.Tests [Runner: JUnit 4] (0.000 s)'. The results indicate 5/5 runs, 0 errors, and 0 failures. A list of tests is shown: testDerivation (0.000 s), testAdd (0.000 s), testMultiplication (0.000 s), testIntegration (0.000 s), and testSubtraction (0.000 s). Below this is a 'Failure Trace' section. On the right, the source code for the 'Tests' class is visible. The code includes imports for Polynomial and assertEquals, and defines several test methods: setUp, testAdd, testSubtraction, testMultiplication, testDivision, testDerivation, and testIntegration. Each test method sets up two polynomials and performs a specific operation, with the result being asserted against an expected string representation.

```
11 public class Tests {
12     Polynomial p1 = new Polynomial();
13     Polynomial p2 = new Polynomial();
14
15     @Before
16     public void setUp() {
17         int[] n1= {3,5,4};
18         int[] n2= {1,3};
19         p1.setPol(n1);
20         p2.setPol(n2);
21     }
22
23     @Test
24     public void testAdd() {
25         Polynomial res = new Polynomial();
26         res = p1.addPolynomials(p2);
27         assertEquals("+3x^3 +6x^2 +7x ", res.toString());
28     }
29
30     @Test
31     public void testSubtraction() {
32         Polynomial res = new Polynomial();
33         res = p1.subtractPolynomials(p2);
34         assertEquals("+3x^3 +4x^2 +x ", res.toString());
35     }
36
37     @Test
38     public void testMultiplication() {
39         Polynomial res = new Polynomial();
40         res = p1.multiplyPolynomials(p2);
41         assertEquals("+3x^5 +14x^4 +19x^3 +12x^2 ", res.toString());
42     }
43
44     @Ignore
45     public void testDivision() {
46
47     }
48
49     @Test
50     public void testDerivation() {
51         p1.derivePolynomial();
52         assertEquals("+9x^2 +10x +4 ", p1.toString());
53     }
54
55     @Test
56     public void testIntegration() {
57         p1.integratePolynomial();
58         assertEquals("+1.0x^4 +2.5x^3 +4.0x^2 ", p1.toStringI());
59     }
60 }
61 }
```

Therefore, the value of p1 is " $3x^3+5x^2+4x$ " and the values of p2 is " x^2+3x ".

By using the function `assertEquals()`, it is shown that for this example the operations are performed correctly.

In the `TestMon` class, the correctness of the operations methods performed on monomials are tested in the same manner:



The screenshot displays an IDE interface. On the left, a 'Runs' tab shows the execution of `mytests.TestsMon` with 4/4 tests passed, 0 errors, and 0 failures. The test methods listed are `testDerivation`, `testMultiplication`, `testIntegration`, and `testDivision`, each taking 0.000 seconds. On the right, the source code for `TestsMon` is shown. It includes the initialization of `Monomial` objects `m1` and `m2`, and four test methods: `testMultiplication`, `testIntegration`, `testDivision`, and `testDerivation`. Each test method uses `assertEquals` to verify the output of the corresponding operation method.

```
9 public class TestsMon {
10     Monomial m1 = new Monomial(-4, 2);
11     Monomial m2 = new Monomial(5, 7);
12
13     @Test
14     public void testMultiplication() {
15         Monomial res = new Monomial(0, 0);
16         res = m1.multiplyMonomials(m2);
17         assertEquals("-20x^9 ", res.toStringM());
18     }
19
20     @Test
21     public void testIntegration() {
22         m1.integrate();
23         assertEquals("-2.0x^3 ", m1.toStringMI());
24     }
25
26     @Test
27     public void testDivision() {
28         Monomial res = new Monomial(0, 0);
29         res = m1.divideMonomials(m2);
30         assertEquals("-0.8x^5 ", res.toStringMI());
31     }
32
33     @Test
34     public void testDerivation() {
35         m1.derivative();
36         assertEquals("-8x ", m1.toStringM());
37     }
38 }
39
40
```

6. Conclusions

The polynomial processing system assignment implementation consisted of creating a user-friendly program which is able to perform basic operations on polynomials. The main parts of this process were implementing the environment in which the operations are performed, designing an interface and linking these two parts together. In addition, it required using unit testing in order to verify the correctness of the method.

7. *Bibliography*

1. <https://stackoverflow.com/questions/40860056/coding-polynomial-division>
2. https://en.wikipedia.org/wiki/Polynomial_arithmetic
3. <http://www.mathcs.emory.edu/~cheung/Courses/377/Syllabus/8-JDBC/GUI/layout.html>