PROGRAMMING TECHNIQUES
ASSIGNMENT 1

SYSTEM FOR POLYNOMIAL PROCESSING

Student: Chindea Miruna
Group: 30424

SUMMARY

1. Homework objective

1.1. Main objective

The main objective of the homework is to propose, design an implement a **system for polynomial processing**. We will consider the polynomials of one variable and integer coefficients.

1.2. Secondary objectives

The secondary objectives of the homework, which will help in building the main objective, are:

1. Creating a **Graphical User Interface** for the system, in order to help the user interact with it. The user will only have to introduce the polynomials, following a pattern and choose the operation he wants to perform by pressing a button, and the result will be automatically displayed if the input is correctly introduced.

This objective will be detailed in Chapter 4.

2. Implementing the following **operations** for the polynomials:

a) addition;

b) subtraction;

c) multiplication;

d) division;

e) integration;

f) differentiation.

We will implement these operations for polynomials, provided that the input will be correctly introduced. Otherwise, the user will not get an eligible answer.

This objective will be detailed in Chapter 4.

3. Providing **Junit Tests**, in order to check the correctness of the implemented operations. We will test all the possible scenarios, from the wrong user input to the case of division by 0.
This objective will be detailed in Chapter 5.

4. Respecting the **OOP Paradigms** in the implementation of the system, e.g. using classes for monomials and polynomials, using *List<>* instead of *array[]*, replacing the classical *for* with *foreach*.

2. Problem analysis, scenarios, use cases

2.1. Functional requirements

There will displayed a graphical interface for the user to interact with. The system should successfully implement the specific operations when the button is pressed, if the user correctly introduces the input. In case the user incorrectly introduces the input, there will be displayed a panel which communicates to the user that the input he introduced is not respecting the pattern. In case the user tries to divide by 0, there will also be displayed a panel saying this kind of operation is forbidden.

2.2. Use cases

The use cases will be presented and described.

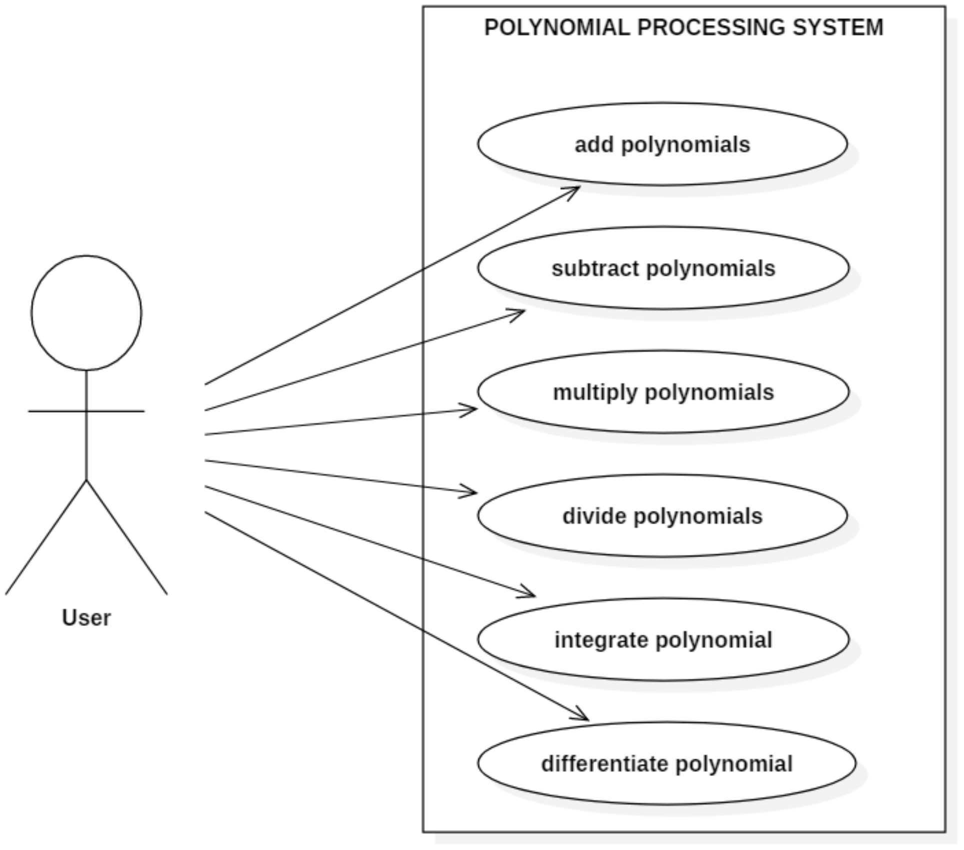


Fig. 1 Use case diagram

- **Use case title: add polynomials**

  - Summary: This use case performs the operation of addition of two polynomials

  - Actors: User

  - Preconditions: The user introduces an input.

  - Main success scenario:

  1. The user introduces the first polynomial
  2. The user introduces the second polynomial
  3. The user presses the "Add polynomials" button
  4. The system takes the first input and checks its correctness
  5. The system takes the second input and checks its correctness
  6. The system performs the operation of addition
  7. The system displays the result

  - Alternative sequences:

  a) First input incorrectly introduced

  1. The system tells the user that the input is incorrect
  2. The scenario returns to step 1

  b) Second input incorrectly introduced

  1. The system tells the user that the input is incorrect
  2. The scenario returns to step 2

  c) The degree of the first polynomial is too high

  1. The system tell the user that the degree is too high
  2. The scenario returns to step 1

  d) The degree of the second polynomial is too high

  1. The system tell the user that the degree is too high
  2. The scenario returns to step 2

  e) The coefficients of the polynomials are not integers

  1. The displayed result will not be correct
  2. The scenario returns to step 1

- **Use case title: subtract polynomials**

  - Summary: This use case performs the operation of subtraction of two polynomials

  - Actors: User

  - Preconditions: The user introduces an input.

  - Main success scenario:

  1. The user introduces the first polynomial
  2. The user introduces the second polynomial
  3. The user presses the "Subtract polynomials" button
  4. The system takes the first input and checks its correctness
  5. The system takes the second input and checks its correctness
  6. The system performs the operation of subtraction
  7. The system displays the result

  - Alternative sequences:

  a) First input incorrectly introduced

  1. The system tells the user that the input is incorrect
  2. The scenario returns to step 1

  b) Second input incorrectly introduced

  1. The system tells the user that the input is incorrect
  2. The scenario returns to step 2

  c) The degree of the first polynomial is too high

  1. The system tell the user that the degree is too high
  2. The scenario returns to step 1

  d) The degree of the second polynomial is too high

  1. The system tell the user that the degree is too high
  2. The scenario returns to step 2

  e) The coefficients of the polynomials are not integers

  1. The displayed result will not be correct
  2. The scenario returns to step 1

- **Use case title: multiply polynomials**

- Summary: This use case performs the operation of multiplication of 2 polynomials

- Actors: User

- Preconditions: The user introduces an input.

- Main success scenario:

1. The user introduces the first polynomial
2. The user introduces the second polynomial
3. The user presses the "Multiply polynomials" button
4. The system takes the first input and checks its correctness
5. The system takes the second input and checks its correctness
6. The system performs the operation of multiplication
7. The system displays the result

- Alternative sequences:

a) First input incorrectly introduced

1. The system tells the user that the input is incorrect
2. The scenario returns to step 1

b) Second input incorrectly introduced

1. The system tells the user that the input is incorrect
2. The scenario returns to step 2

c) The degree of the first polynomial is too high

1. The system tell the user that the degree is too high
2. The scenario returns to step 1

d) The degree of the second polynomial is too high

1. The system tell the user that the degree is too high
2. The scenario returns to step 2

e) The resulted polynomial's degree is too high

1. The displayed result will not be correct
2. The scenario returns to step 1

f) The coefficients of the polynomials are not integers

1. The displayed result will not be correct
2. The scenario returns to step 1

- **Use case title: divide polynomials**

  - Summary: This use case performs the operation of division of two polynomials

  - Actors: User

  - Preconditions: The user introduces an input.

  - Main success scenario:

  1. The user introduces the first polynomial
  2. The user introduces the second polynomial
  3. The user presses the “Divide polynomials” button
  4. The system takes the first input and checks its correctness
  5. The system takes the second input and checks its correctness
  6. The system performs the operation of division
  7. The system displays the result

  - Alternative sequences:

  a) First input incorrectly introduced

  1. The system tells the user that the input is incorrect
  2. The scenario returns to step 1

  b) Second input incorrectly introduced

  1. The system tells the user that the input is incorrect
  2. The scenario returns to step 2

  c) The degree of the first polynomial is too high

  1. The system tell the user that the degree is too high
  2. The scenario returns to step 1

  d) The degree of the second polynomial is too high

  1. The system tell the user that the degree is too high
  2. The scenario returns to step 2

  e) The coefficients of the polynomials are not integers

  1. The displayed result will not be correct
  2. The scenario returns to step 1

  f) The second polynomial is equal to 0

  1. The system tells the user he cannot divide by 0
  2. The scenario returns to step 2

- **Use case title: integrate polynomial**

  - Summary: This use case performs the operation of integration of a polynomial

  - Actors: User

  - Preconditions: The user introduces an input.

  - Main success scenario:

  1. The user introduces the polynomial
  2. The user presses the "Integrate polynomial" button
  3. The system takes the input and checks its correctness
  4. The system performs the operation of integration
  5. The system displays the result

  - Alternative sequences:

  a) Input incorrectly introduced

  1. The system tells the user that the input is incorrect
  2. The scenario returns to step 1

  b) The degree of the polynomial is too high

  1. The system tell the user that the degree is too high
  2. The scenario returns to step 1

  c) The coefficient of the polynomial is not an integer

  1. The displayed result will not be correct
  2. The scenario returns to step 1

- **Use case title: differentiate polynomial**

  - Summary: This use case performs the operation of differentiation of a polynomial

  - Actors: User

  - Preconditions: The user introduces an input.

  - Main success scenario:

  1. The user introduces the polynomial
  2. The user presses the "Differentiate polynomial" button
  3. The system takes the input and checks its correctness
  4. The system performs the operation of differentiation
  5. The system displays the result

- Alternative sequences:

a) Input incorrectly introduced

1. The system tells the user that the input is incorrect
2. The scenario returns to step 1

b) The degree of the polynomial is too high

1. The system tell the user that the degree is too high
2. The scenario returns to step 1

c) The coefficient of the polynomial is not an integer

1. The displayed result will not be correct
2. The scenario returns to step 1

3. Design

3.1. Pattern

The design is based on the Model-View-Controller pattern. It divides the application into three interconnected parts, in order to separate the internal representations of information from the ways information is presented to and accepted from the user. The components are presented below:

3.1.1. Model

The model is the central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application. The model is responsible for managing the data of the application. It receives user input from the controller.

3.1.2. View

The view means presentation of the model in a particular format, Any representation of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.

3.1.3. Controller

The controller responds to the user input and performs interactions on the data model objects. It receives the input, optionally validates it and then passes the input to the model.

3.2. Classes and packages

The classes structured in packages according to the MVC pattern. Using these packages helps in organizing the classes and interfaces. The packages and classes will be presented on the next page. The classes and packages structure is illustrated in Fig. 1.

1. model

This is where the input is processed and where all the operations are performed. This package contains the following classes:

a) Monomial: is represented by a degree and a coefficient

b) Polynomial: is structured as an *ArrayList*<> of monomials and has a degree

c) Operations: here are all the operations on polynomials implemented

d) DegreeComparator: this class implements the Comparator interface and is used to sort the monomials in the polynomial by their degree

e) Model: this class is used for interconnection with the View and the Controller

2. view

This package contains the View class, which implements the Action Listener interface. This is where the Graphical User Interface is created, without any functionality implemented. It contains frames, panels, text fields, buttons and labels.

3. controller

This package contains the Controller class, which creates a functionality to the View, by connecting it with the Model.

4. mvc

This package contains the MVC class, which interconnects the three classes, Model, View and Controller and offers a functionality to the whole application.

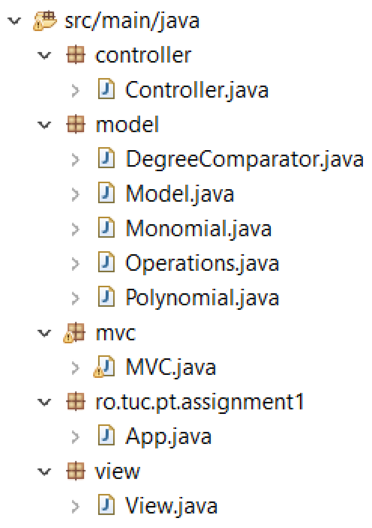

src/main/java
controller
Controller.java
model
DegreeComparator.java
Model.java
Monomial.java
Operations.java
Polynomial.java
mvc
MVC.java
ro.tuc.pt.assignment1
App.java
view
View.java

Fig. 2 Packages and classes

3.3. Interfaces

The interfaces that were implemented in this application are:

- *ActionListener:* implemented by the View and Controller classes. This interface is used for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListenermethod. When the action event occurs, that object's actionPerformed method is invoked.

- *Comparator:* implemented by the DegreeComparator class. This interface is used in this application in order to sort the monomials in a polynomial by their degree. This interface uses a comparison function, which imposes a total ordering on some collection of objects.

3.4. UML Diagram

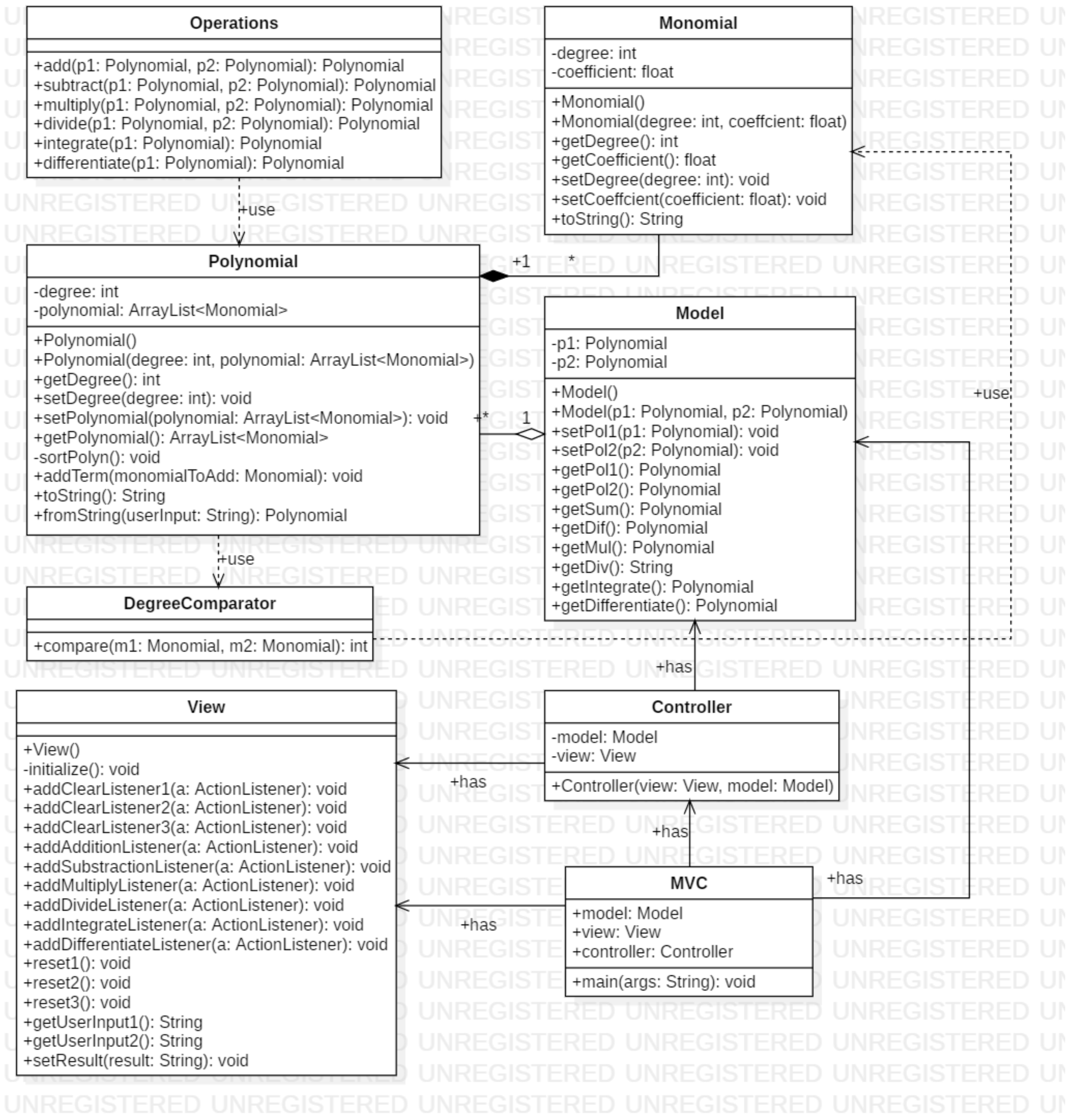

Operations
+add(p1: Polynomial, p2: Polynomial): Polynomial
+subtract(p1: Polynomial, p2: Polynomial): Polynomial
+multiply(p1: Polynomial, p2: Polynomial): Polynomial
+divide(p1: Polynomial, p2: Polynomial): Polynomial
+integrate(p1: Polynomial): Polynomial
+differentiate(p1: Polynomial): Polynomial
Monomial
-degree: int
-coefficient: float
+Monomial()
+Monomial(degree: int, coeffcient: float)
+getDegree(): int
+getCoefficient(): float
+setDegree(degree: int): void
+setCoeffcient(coefficient: float): void
+toString(): String
+use
+1
*
Polynomial
-degree: int
-polynomial: ArrayList<Monomial>
+Polynomial()
+Polynomial(degree: int, polynomial: ArrayList<Monomial>)
+getDegree(): int
+setDegree(degree: int): void
+setPolynomial(polynomial: ArrayList<Monomial>): void
+getPolynomial(): ArrayList<Monomial>
-sortPolyn(): void
+addTerm(monomialToAdd: Monomial): void
+toString(): String
+fromString(userInput: String): Polynomial
+*
1
Model
-p1: Polynomial
-p2: Polynomial
+Model()
+Model(p1: Polynomial, p2: Polynomial)
+setPol1(p1: Polynomial): void
+setPol2(p2: Polynomial): void
+getPol1(): Polynomial
+getPol2(): Polynomial
+getSum(): Polynomial
+getDif(): Polynomial
+getMul(): Polynomial
+getDiv(): String
+getIntegrate(): Polynomial
+getDifferentiate(): Polynomial
+use
+use
DegreeComparator
+compare(m1: Monomial, m2: Monomial): int
+has
View
+View()
-initialize(): void
+addClearListener1(a: ActionListener): void
+addClearListener2(a: ActionListener): void
+addClearListener3(a: ActionListener): void
+addAdditionListener(a: ActionListener): void
+addSubstractionListener(a: ActionListener): void
+addMultiplyListener(a: ActionListener): void
+addDivideListener(a: ActionListener): void
+addIntegrateListener(a: ActionListener): void
+addDifferentiateListener(a: ActionListener): void
+reset1(): void
+reset2(): void
+reset3(): void
+getUserInput1(): String
+getUserInput2(): String
+setResult(result: String): void
Controller
-model: Model
-view: View
+Controller(view: View, model: Model)
+has
+has
+has
MVC
+model: Model
+view: View
+controller: Controller
+main(args: String): void
+has

Fig. 3 UML Diagram

3.5. Relationships

- Monomial-Polynomial: there is a **Composition** relationship between these two classes. The polynomial cannot exist alone without a monomial, neither does a monomial without a polynomial to be part of. These two classes are dependent on each other and their life span are the same.

- Polynomial-Operations: there is a **Dependency** relationship between these two classes. The class Operations "uses" the polynomials in order to perform the operations. The class Operation uses one or two instances of the Polynomial class in its methods.

- **Polynomial-Model:** there is an **Aggregation** relationship between these two classes. The Model class contains two Polynomial entities.

- Polynomial-DegreeComparator: there is a **Dependency** relationship between these two classes. The class Polynomial uses an instance of the DegreeComparator class in a method.

- Monomial-DegreeComparator: there is a **Dependency** relationship between these two classes. The class DegreeComparator uses two instances of the Monomial class as arguments of a method.

- Model-Controller: there is an **Association** relationship between these two classes. The Controller class has as attribute an instance of the Model class.

- View-Controller: there is an **Association** relationship between these two classes. The Controller class has as attribute an instance of the View class.

- **Model-MVC:** there is an **Association** relationship between these two classes. The MVC class has as attribute an instance of the Model class.

- **View-MVC:** there is an **Association** relationship between these two classes. The MVC class has as attribute an instance of the View class.

- **Controller-MVC:** there is an **Association** relationship between these two classes. The MVC class has as attribute an instance of the Controller class.

4. Implementation

In this chapter, there will be rigorously described:

- all the classes with their attributes and important methods;

- the Graphical User Interface.

4.1. Classes

4.1.1. Monomial

Represents the monomial entity. It has two constructors, one parameterless and one with degree and coefficient as parameters. The Monomial class is in a Composition relationship with the Polynomial class, meaning that a monomial cannot exist if it is not part of a polynomial.

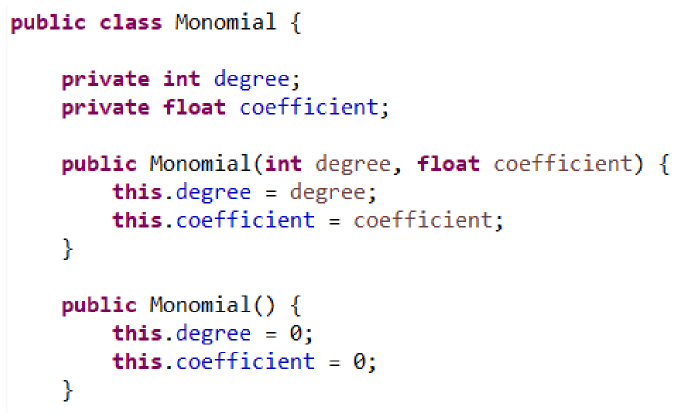
Attributes:

- *degree: int* – the degree of the monomial;

- *coefficient: float* – the coefficient of the monomial.

Methods:

- *toString(): String* – returns a String representing the monomial in an user-readable manner (e.g. 2x^3).



```
public class Monomial {

    private int degree;
    private float coefficient;

    public Monomial(int degree, float coefficient) {
        this.degree = degree;
        this.coefficient = coefficient;
    }

    public Monomial() {
        this.degree = 0;
        this.coefficient = 0;
    }
```

Fig. 4 Monomial class with constructors

4.1.2. Polynomial

Represents the polynomial entity. It has two constructors, one parameterless and one having a degree and an ArrayList<> of Monomials as parameters. This class is in a Composition relationship with the Monomial class, meaning that a polynomial cannot exist without a monomial. The class is also in a Dependency relationship with the Operations class, which implements the operations on the polynomials.

Attributes:

- *degree: int* – the degree of the polynomial is the highest degree from the monomials that it contains;

- *polynomial: ArrayList<Monomial>* – a list containing all the monomial entities of a polynomial.

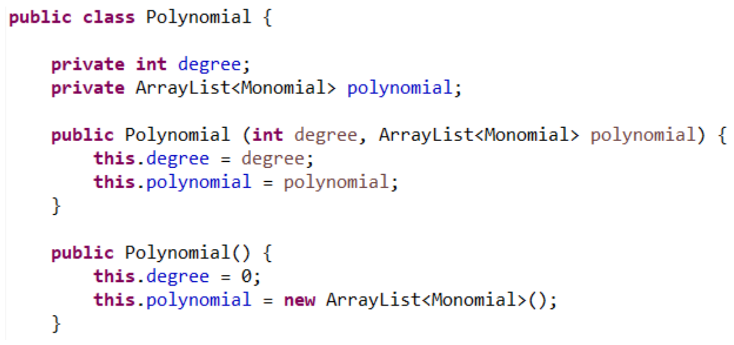
Methods:

- *sortPolyn(): void* – sorts the monomials in the polynomials in a descending order according to their degree;

- *addTerm(monomialToAdd: Monomial): void* – adds the monomial from the argument in the polynomial list; if there is already a monomial with the same degree in the lists, only its coefficient is added to the existing monomial;

- *toString(): String* – returns a String representing the polynomial in an user- readable manner(e.g. 2x^2 + x + 3);

- *fromString(String userInput): Polynomial* – takes as argument a String representing a polynomial in an user-written manner and computes the corresponding Polynomial. The userInput argument is the input that is introduced by the user in the graphical interface.



```
public class Polynomial {

    private int degree;
    private ArrayList<Monomial> polynomial;

    public Polynomial (int degree, ArrayList<Monomial> polynomial) {
        this.degree = degree;
        this.polynomial = polynomial;
    }

    public Polynomial() {
        this.degree = 0;
        this.polynomial = new ArrayList<Monomial>();
    }
```

Fig. 5 Polynomial class with constructors

4.1.3. Model

This class is the Model component from the MVC pattern design. It has as attributes two polynomials. In the methods, the operations from the Operation class are called and the result is returned. It has two constructors, one parameterless and one having as arguments two poylinomials.

Attributes:

- *p1: Polynomial* – the first polynomial

- *p2: Polynomial* – the second polynomial

Methods:

- *getSum(): Polynomial* – returns the result of addition of two polynomials

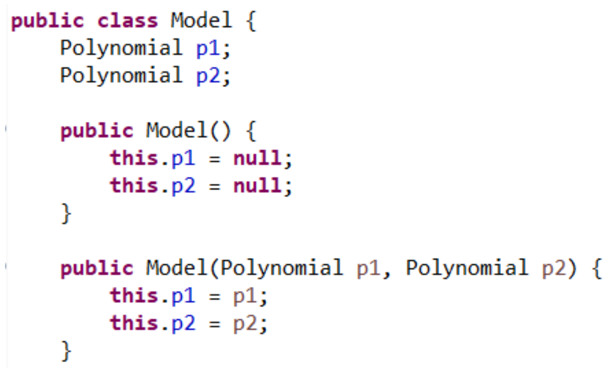
- *getDif(): Polynomial* – returns the result of subtraction of two polynomials

- *getMul(): Polynomial* – returns the result of multiplication of two polynomials

- *getDiv(): String* – returns the result of division of two polynomials as a String containing the quotient and the remainder of the division

- *getIntegrate(): Polynomial* – returns the result of integration of the first polynomial

- *getDifferentiate(): Polynomial* – returns the result of differentiation of the first polynomial



```java
public class Model {
    Polynomial p1;
    Polynomial p2;

    public Model() {
        this.p1 = null;
        this.p2 = null;
    }

    public Model(Polynomial p1, Polynomial p2) {
        this.p1 = p1;
        this.p2 = p2;
    }
```

Fig. 6 Model class with constructors

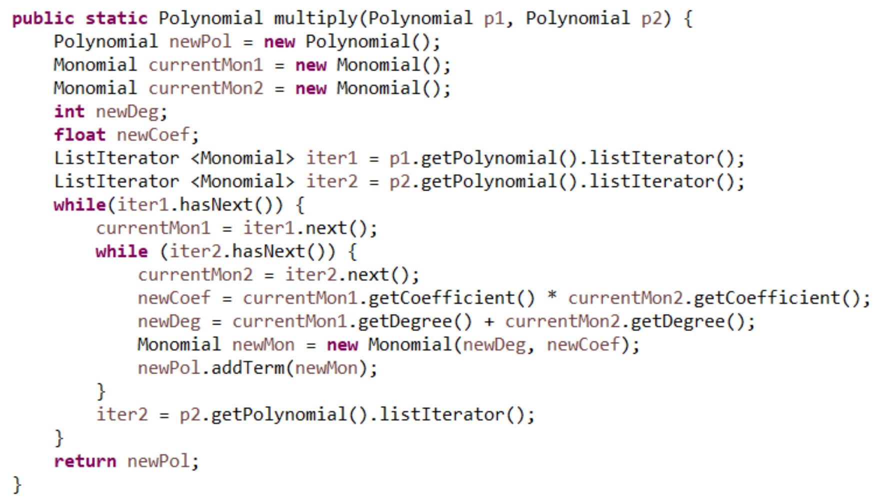
4.1.4. Operations

This is where the operations on polynomials are implemented. In every operation, in order to access the monomials contained in a polynomial, we use an iterator which goes all the way through the polynomial and performs the operations. This class has no attributes, all its methods are static.

Methods:

- *add(p1: Polynomial, p2: Polynomial): Polynomial* – performs the operation of addition of the two polynomials in the parameter list; we iterate through the two polynomials and add their monomials to a new polynomials, calling the *addTerm*() method;

- *subtract(p1: Polynomial, p2: Polynomial): Polynomial* – performs the operation of subtraction of the two polynomials in the parameter list; we perform the operation same as the addition, only changing the signs of second's polynomial monomials;
- *multiply(p1: Polynomial, p2: Polynomial): Polynomial* – performs the operation of multiplication of the two polynomials in the parameter list; we perform the operation of multiplication of every monomial of p1 with every monomial of p2;
- *divide(p1: Polynomial, p2: Polynomial): Polynomial[]* – performs the operation of addition of the two polynomials in the parameter list; returns an array of two polynomials representing the quotient and the remainder; we used Horner's method for dividing the polynomials in implementing this method;
- *integrate(p1: Polynomial): Polynomial* – performs the operation of integration of a polynomial; we perform this operation directly on the polynomial, incrementing the *degree* and computing a new *coefficient;*
- *differentiate(p1: Polynomial): Polynomial* – performs the operation of differentiation of a polynomial; we perform this operation directly on the polynomial, decrementing the *degree* and computing a new *coefficient;*

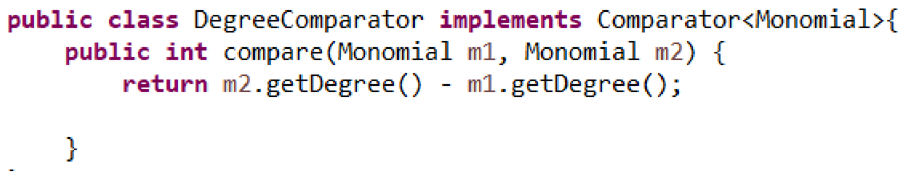


```java
public static Polynomial multiply(Polynomial p1, Polynomial p2) {
    Polynomial newPol = new Polynomial();
    Monomial currentMon1 = new Monomial();
    Monomial currentMon2 = new Monomial();
    int newDeg;
    float newCoef;
    ListIterator <Monomial> iter1 = p1.getPolynomial().listIterator();
    ListIterator <Monomial> iter2 = p2.getPolynomial().listIterator();
    while(iter1.hasNext()) {
        currentMon1 = iter1.next();
        while (iter2.hasNext()) {
            currentMon2 = iter2.next();
            newCoef = currentMon1.getCoefficient() * currentMon2.getCoefficient();
            newDeg = currentMon1.getDegree() + currentMon2.getDegree();
            Monomial newMon = new Monomial(newDeg, newCoef);
            newPol.addTerm(newMon);
        }
        iter2 = p2.getPolynomial().listIterator();
    }
    return newPol;
}
```

Fig. 7 Multiplication method from Operations class

4.1.5. DegreeComparator

This class implements the Comparator interface. It has only one method, *compare(m1: Monomial, m2: Monomial):int* , which compares the degree of two monomials from the parameter list.



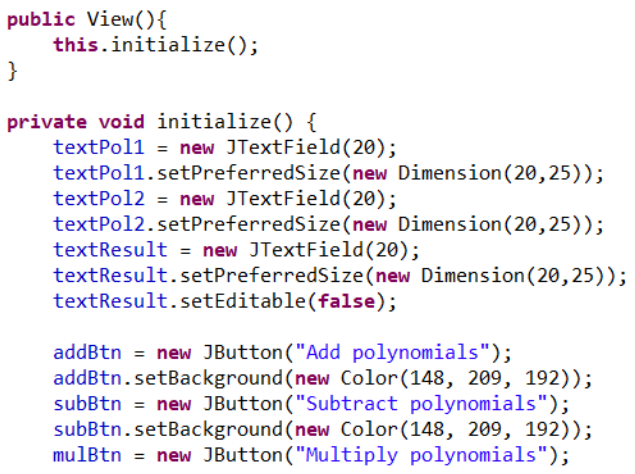
```
public class DegreeComparator implements Comparator<Monomial>{
    public int compare(Monomial m1, Monomial m2) {
        return m2.getDegree() - m1.getDegree();

    }
```

Fig. 8 DegreeComparator class

4.2. Graphical User Interface

4.2.1. View

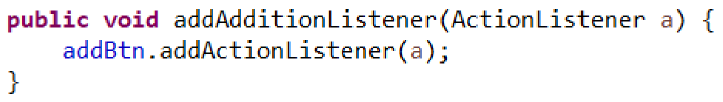
This class is the View component from the MVC pattern design. Here we instantiate the frames, panels, buttons, text fields and labels. The constructor of the class calls the *initialize(): void* method, which instantiates all the components.



```
public View(){
    this.initialize();
}

private void initialize() {
    textPol1 = new JTextField(20);
    textPol1.setPreferredSize(new Dimension(20,25));
    textPol2 = new JTextField(20);
    textPol2.setPreferredSize(new Dimension(20,25));
    textResult = new JTextField(20);
    textResult.setPreferredSize(new Dimension(20,25));
    textResult.setEditable(false);

    addBtn = new JButton("Add polynomials");
    addBtn.setBackground(new Color(148, 209, 192));
    subBtn = new JButton("Subtract polynomials");
    subBtn.setBackground(new Color(148, 209, 192));
    mulBtn = new JButton("Multiply polynomials");
```

Fig. 9 View class with constructor and initialize() method

Other methods from this class are used to add action listeners to the buttons.

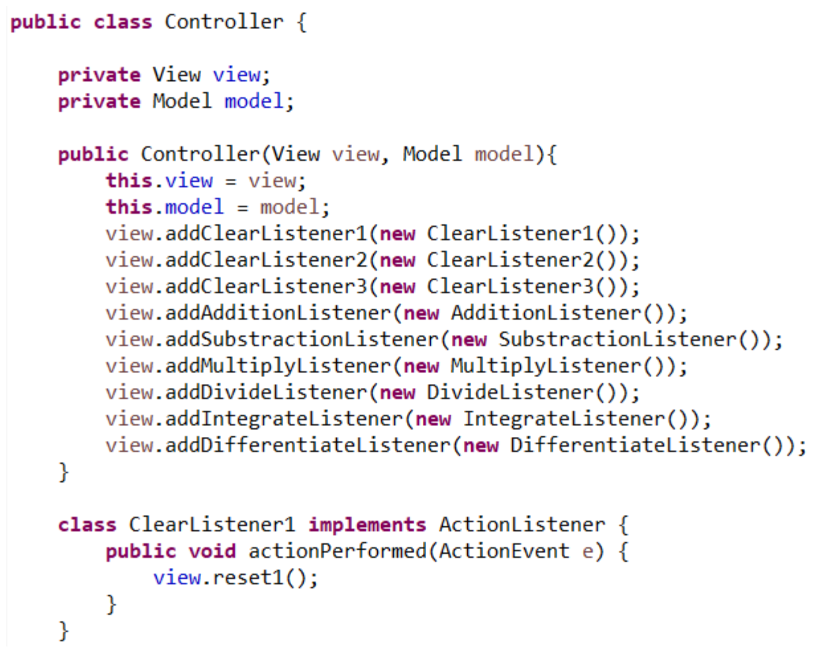


```
public void addAdditionListener(ActionListener a) {
    addBtn.addActionListener(a);
}
```

Fig. 10 Method from View class

4.2.2. Controller

This class interconnects the Model and View classes and offers a functionality to the Graphical User Interface. It has as attributes a Model entity and a View entity and contains classes that implement the ActionListener interface. We have classes of Listeners for every button from the View class.

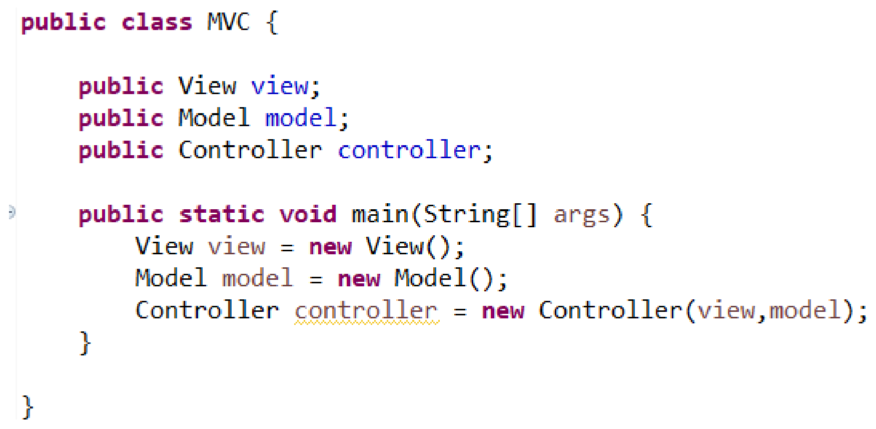


```
public class Controller {

    private View view;
    private Model model;

    public Controller(View view, Model model){
        this.view = view;
        this.model = model;
        view.addClearListener1(new ClearListener1());
        view.addClearListener2(new ClearListener2());
        view.addClearListener3(new ClearListener3());
        view.addAdditionListener(new AdditionListener());
        view.addSubstractionListener(new SubstractionListener());
        view.addMultiplyListener(new MultiplyListener());
        view.addDivideListener(new DivideListener());
        view.addIntegrateListener(new IntegrateListener());
        view.addDifferentiateListener(new DifferentiateListener());
    }

    class ClearListener1 implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            view.reset1();
        }
    }
```

Fig. 11 Controller class with constructor and Listener class

4.2.3. MVC

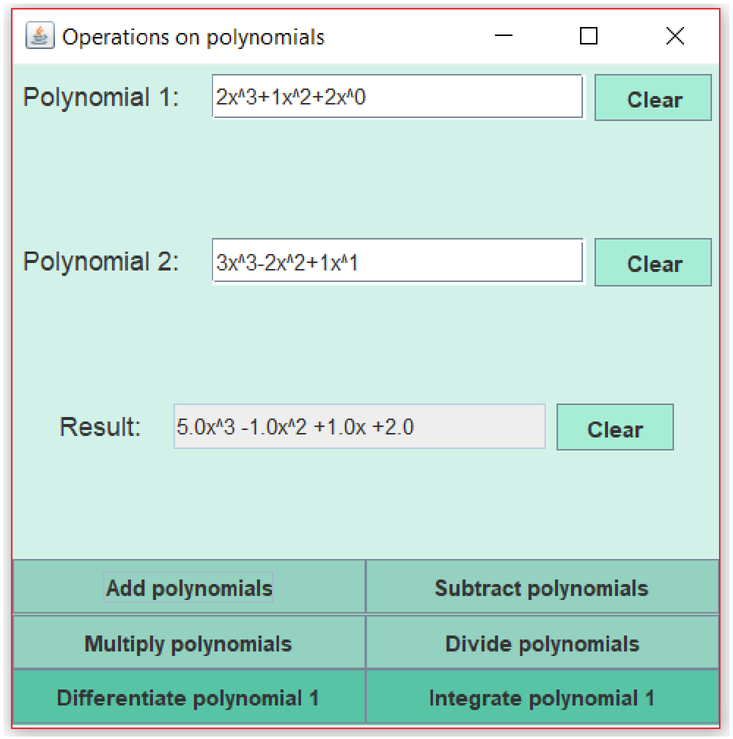
This is the class where we interconnect all the components from the Model-View-Controller pattern design. The application is ready for launching now.



```
public class MVC {

    public View view;
    public Model model;
    public Controller controller;

    public static void main(String[] args) {
        View view = new View();
        Model model = new Model();
        Controller controller = new Controller(view,model);
    }

}
```

Fig. 12 Model-View-Controller class

When we run the MVC class, the Graphical User Interface will be displayed and the user can perform the wished operation.


Operations on polynomials

Polynomial 1: 2x^3+1x^2+2x^0

Polynomial 2: 3x^3-2x^2+1x^1

Result: 5.0x^3 -1.0x^2 +1.0x +2.0

Fig. 13 User Interface after the addition operation

5. Results

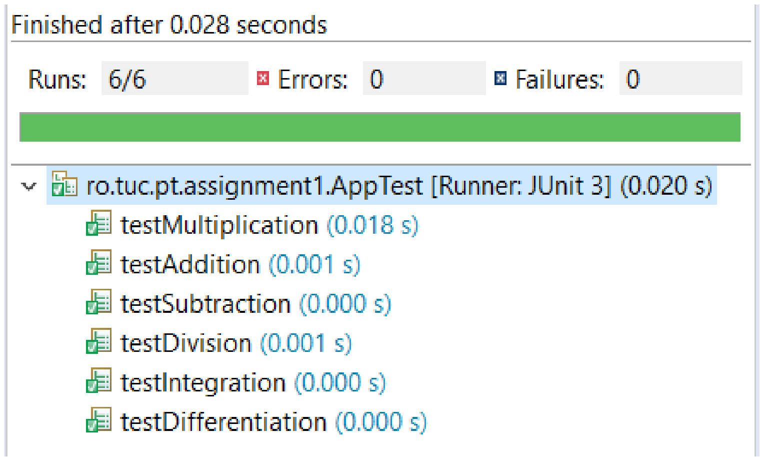
In this chapter, the testing scenarios will be presented. JUnit Tests will be performed on different inputs.

1. Correctly introduced inputs

```
public void setUp() {
    p1 = Polynomial.fromString("2x^3+7x^2+2x^1+9x^0");
    p2 = Polynomial.fromString("2x^1+3x^0");
}
```

Fig. 14 Correctly introduced input for the 1st test

The results are correct, no errors encountered.



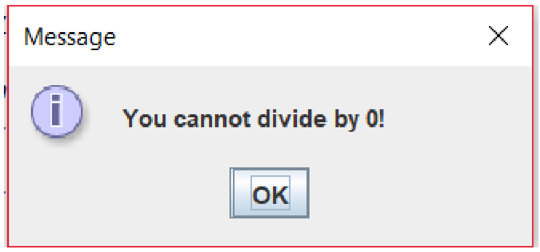
```
Finished after 0.028 seconds

Runs: 6/6    Errors: 0    Failures: 0

ro.tuc.pt.assignment1.AppTest [Runner: JUnit 3] (0.020 s)
    testMultiplication (0.018 s)
    testAddition (0.001 s)
    testSubtraction (0.000 s)
    testDivision (0.001 s)
    testIntegration (0.000 s)
    testDifferentiation (0.000 s)
```

Fig. 15 Results of the 1st test

2. Division by 0: If the user tries to divide by 0, a panel will be displayed telling the user the operation is forbidden and the result will be by default 0.

```
    public void testDivisionByZero()
  {
      p2 = Polynomial.fromString("0x^0");
      result1 = Operations.divide(p1, p2)[0];
      s1 = result1.toString();
      result2 = Operations.divide(p1, p2)[1];
      s2 = result2.toString();
      s3 = "0.0 ";
      s4 = "0.0 ";
      assertTrue(s1.equals(s3) && s2.equals(s4));
  }
```

Fig. 16 Testing division by 0



Message

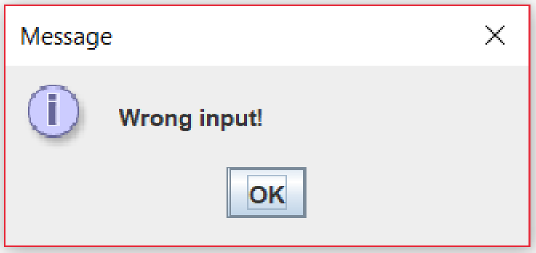
You cannot divide by 0!

Fig. 17 System warning the user that division by 0 is forbidden

3. Wrong input: If the user doesn't introduce a correct input, the system will tell the user about the problem and, in the following case, the result will take the value of the first polynomial, which was correctly introduced.

```
public void testWrongInput() {
    p2 = Polynomial.fromString("abcde");
    result1 = Operations.add(p1, p2);
    s1 = result1.toString();
    s2 = s1.toString();
    assertTrue(s1.equals(s2));
}
```

Fig. 18 Testing incorrect input


Message

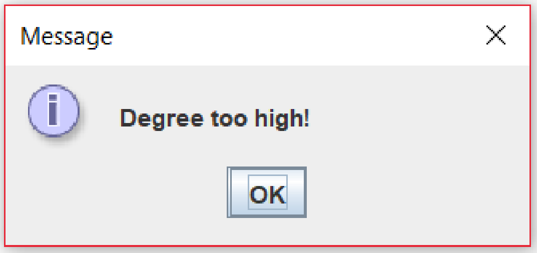
Wrong input!

Fig. 19 System telling the input is wrong

4. Numbers too large: If the degree of a polynomial is too high, the system will warn the user and the operation will not correctly perform. In this case, the addition will take the value of the first polynomial, the degree of the second one being too large.

```
public void testHighDegree() {
    p2 = Polynomial.fromString("3x^9289876327387");
    result1 = Operations.add(p1, p2);
    s1 = result1.toString();
    s2 = s1.toString();
    assertTrue(s1.equals(s2));
}
```

Fig. 20 Testing high degree of polynomial


Message

Degree too high!

Fig. 21 System telling the degree is too large

The previously presented scenarios were correctly evaluated by the JUnit Test.

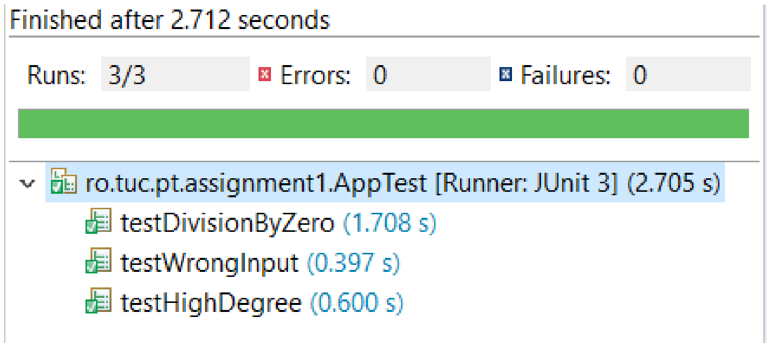

```
Finished after 2.712 seconds

Runs: 3/3    Errors: 0    Failures: 0

ro.tuc.pt.assignment1.AppTest [Runner: JUnit 3] (2.705 s)
    testDivisionByZero (1.708 s)
    testWrongInput (0.397 s)
    testHighDegree (0.600 s)
```

Fig. 22 Test of the scenarios

6. Conclusions

In conclusion, the system successfully computes the operations if the input is introduced correctly. If the user incorrectly introduces the polynomial, the system will display message dialogs, communicating the user the mistake, and the user can modify the input into a correct one, respecting the pattern.

I learnt from this project more information about OOP Paradigms, practiced my code writing and learnt about JUnit Tests and Use-case diagrams.

This application can be improved in many ways, for example:

- treating all the cases for the incorrectly introduced input;
- increasing the maximum allowed degree and coefficient;
- improving the GUI into a more complex and user-pleasing one;

- allowing the coefficients to be real numbers, not only integers;
- adding an operation to find the roots of the polynomial and other operations.

7. Bibliography

https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

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

http://zetcode.com/tutorials/javaswingtutorial/

https://www.tutorialspoint.com/junit/index.htm