Fundamental Programming Techniques 2021

Assignment 1

Polynomial calculator

Java application

Chereji Iulia-Adela

Group 30424-1

CONTENTS

1. ASSIGNMENT OBJECTIVE

1.1 Main objective

Design and implement a polynomial calculator with a dedicated graphical interface through which the user can insert polynomials, select the mathematical operation (i.e. addition, subtraction, multiplication, division, derivative, integration) to be performed and view the result. The polynomials have integer coefficients and one variable.

1.2 Secondary objectives

1.2.1 Analyze the problem and identify requirements

The modeling of the problem will be performed (i.e. identify the functional and non-functional requirements, identify the scenarios, detail and analyze the use cases). – Ch. 2

1.2.2 Design the polynomial calculator

The OOP design decisions will be presented (i.e. UML diagrams, data structures, class design, relationships, packages, user interfaces). –Ch. 3

1.2.3 Implement the polynomial calculator

The implementation of the classes (i.e. fields and important methods) and of the graphical user interface will be presented. –Ch. 4

1.2.4 Test the polynomial calculator

The testing scenarios that were performed on the application using Junit will be described. –Ch. 5

2. PROBLEM ANALYSIS

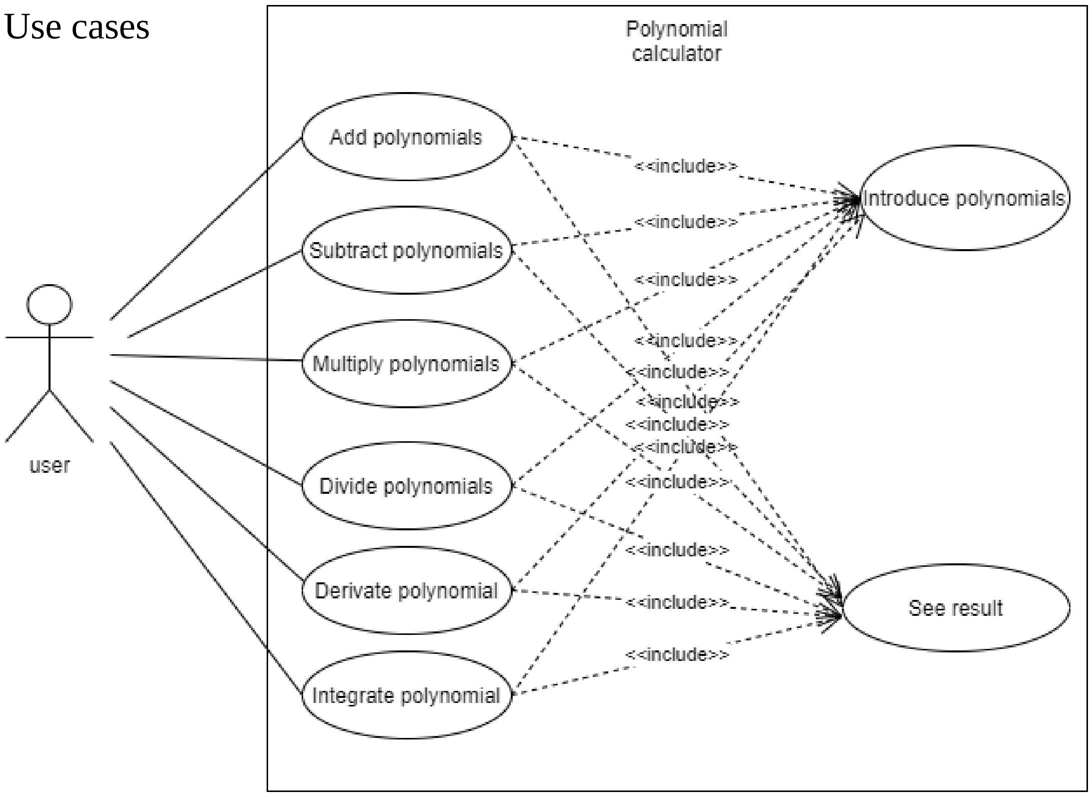
2.1 Functional requirements

- The polynomial calculator should allow users to insert polynomials
- The polynomial calculator should allow users to select the mathematical operation
- The polynomial calculator should add two polynomials
- The polynomial calculator should subtract the second polynomial from the first
- The polynomial calculator should multiply two polynomials
- The polynomial calculator should divide the first polynomial to the second
- The polynomial calculator should derivate a polynomial
- The polynomial calculator should integrate a polynomial
- The polynomial calculator should display the result

2.2 Non-functional requirements

- The polynomial calculator should be intuitive and easy to use by the user
- The polynomial calculator should display a warning if the user's input is incorrect (wrong format; first polynomial not introduced; second polynomial not introduced in the case of addition, subtraction, multiplication and division; division by zero).
- The polynomial calculator should give the user an example of a correct input.
- The polynomial calculator should expect and return the polynomials in a readable and understandable manner (+, -, ^, spaces).

2.3 Use cases



2.3.1 Use Case: Add polynomials

Primary Actor: user

Main Success Scenario:

1. The user inserts the 2 polynomials in the graphical user interface.
2. The user clicks on the “ADD” button.
3. The polynomial calculator performs the addition of the two polynomials and displays the result.

Alternative Sequence: Incorrect polynomials

1. The user inserts incorrect polynomials (e.g. only one or no polynomial, with 2 or more variables, without spaces, with non-accepted characters, with non-integer coefficients or degrees).
2. The user clicks on the “ADD” button.
3. The polynomial calculator displays an error message.
4. The scenario returns to step 1.

2.3.2 Use Case: Subtract polynomials

Primary Actor: user

Main Success Scenario:

1. The user inserts the 2 polynomials in the graphical user interface.
2. The user clicks on the “SUBTRACT” button.
3. The polynomial calculator performs the subtraction of the second polynomial from the first polynomial and displays the result.

Alternative Sequence: Incorrect polynomials

1. The user inserts incorrect polynomials (e.g. only one or no polynomial, with 2 or more variables, without spaces, with non-accepted characters, with non-integer coefficients or degrees).
2. The user clicks on the “SUBTRACT” button.
3. The polynomial calculator displays an error message.
4. The scenario returns to step 1.

2.3.3 Use Case: Multiply polynomials

Primary Actor: user

Main Success Scenario:

1. The user inserts the 2 polynomials in the graphical user interface.
2. The user clicks on the “MULTIPLY” button.
3. The polynomial calculator performs the multiplication of the two polynomials and displays the result.

Alternative Sequence: Incorrect polynomials

1. The user inserts incorrect polynomials (e.g. only one or no polynomial, with 2 or more variables, without spaces, with non-accepted characters, with non-integer coefficients or degrees).
2. The user clicks on the “MULTIPLY” button.
3. The polynomial calculator displays an error message.
4. The scenario returns to step 1.

2.3.4 Use Case: Divide polynomials

Primary Actor: user

Main Success Scenario:

1. The user inserts the 2 polynomials in the graphical user interface.
2. The user clicks on the “DIVIDE” button.
3. The polynomial calculator performs division of the first polynomial to the second polynomial and displays the result.

Alternative Sequence: Incorrect polynomials

1. The user inserts incorrect polynomials (e.g. only one or no polynomial, with 2 or more variables, without spaces, with non-accepted characters, with non-integer coefficients or degrees, with the second polynomial being zero).
2. The user clicks on the “DIVIDE” button.
3. The polynomial calculator displays an error message.
4. The scenario returns to step 1.

2.3.5 Use Case: Derivate polynomial

Primary Actor: user

Main Success Scenario:

1. The user inserts the first polynomial in the graphical user interface.
2. The user clicks on the “DERIVATE” button.
3. The polynomial calculator performs the derivation of the first polynomial and displays the result.

Alternative Sequence: Incorrect polynomial

1. The user inserts an incorrect polynomial (e.g. first polynomial null, with 2 or more variables, without spaces, with non-accepted characters, with non-integer coefficients or degrees).
2. The user clicks on the “DERIVATE” button.
3. The polynomial calculator displays an error message.
4. The scenario returns to step 1.

2.3.6 Use Case: Integrate polynomial

Primary Actor: user

Main Success Scenario:

1. The user inserts the first polynomial in the graphical user interface.

2. The user clicks on the “INTEGRATE” button.
3. The polynomial calculator performs the integration of the first polynomial and displays the result.

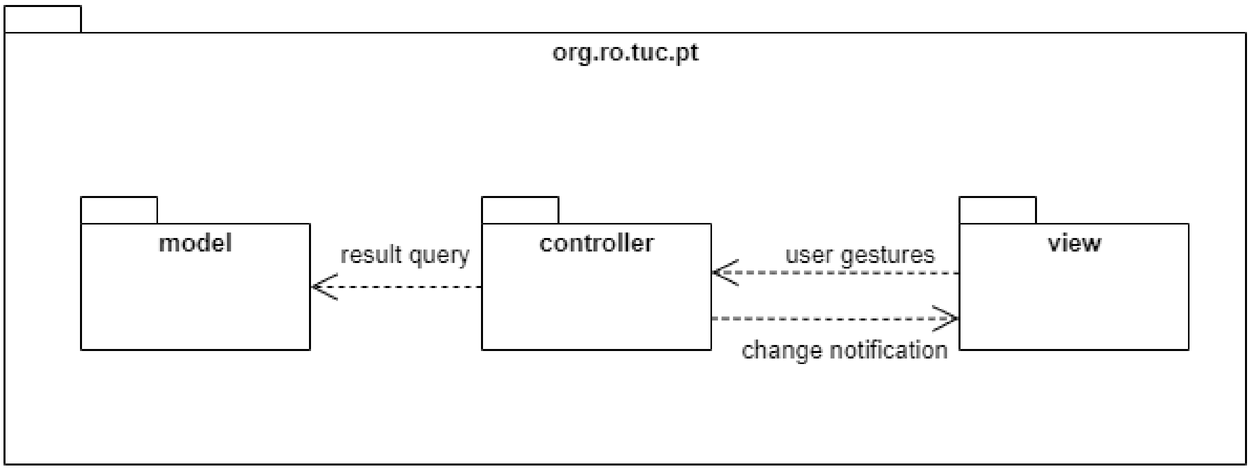
Alternative Sequence: Incorrect polynomial

1. The user inserts an incorrect polynomial (e.g. first polynomial null, with 2 or more variables, without spaces, with non-accepted characters, with non-integer coefficients or degrees).
2. The user clicks on the “INTEGRATE” button.
3. The polynomial calculator displays an error message.
4. The scenario returns to step 1.

3. DESIGN DECISIONS

3.1 Division into packages

The Model View Controller Architectural Pattern was used.



- model package encapsulates the functionality and the data models.
- view package receives input from the user and displays the results given by the controller.
- controller package handles the user requests, sends data to be processed to the model and decides what the view must do as a consequence of the model's results.

3.2 Division into classes

3.2.1 CRC cards

App	
• Creates the View object • Creates the Controller object	• View • Controller

Controller	
• Adds Button Listeners • Adds Field Listeners • Processes input • Decides what to be displayed	• View • ButtonsListener • FieldListener • MyRunnable • Operations

View	JFrame
• Takes input • Displays output and messages	• Controller

FieldListener	
• Reads input	• View • Controller

ButtonsListener	
• Gets the pressed button • Starts the processing	• Controller • View • MyRunnable

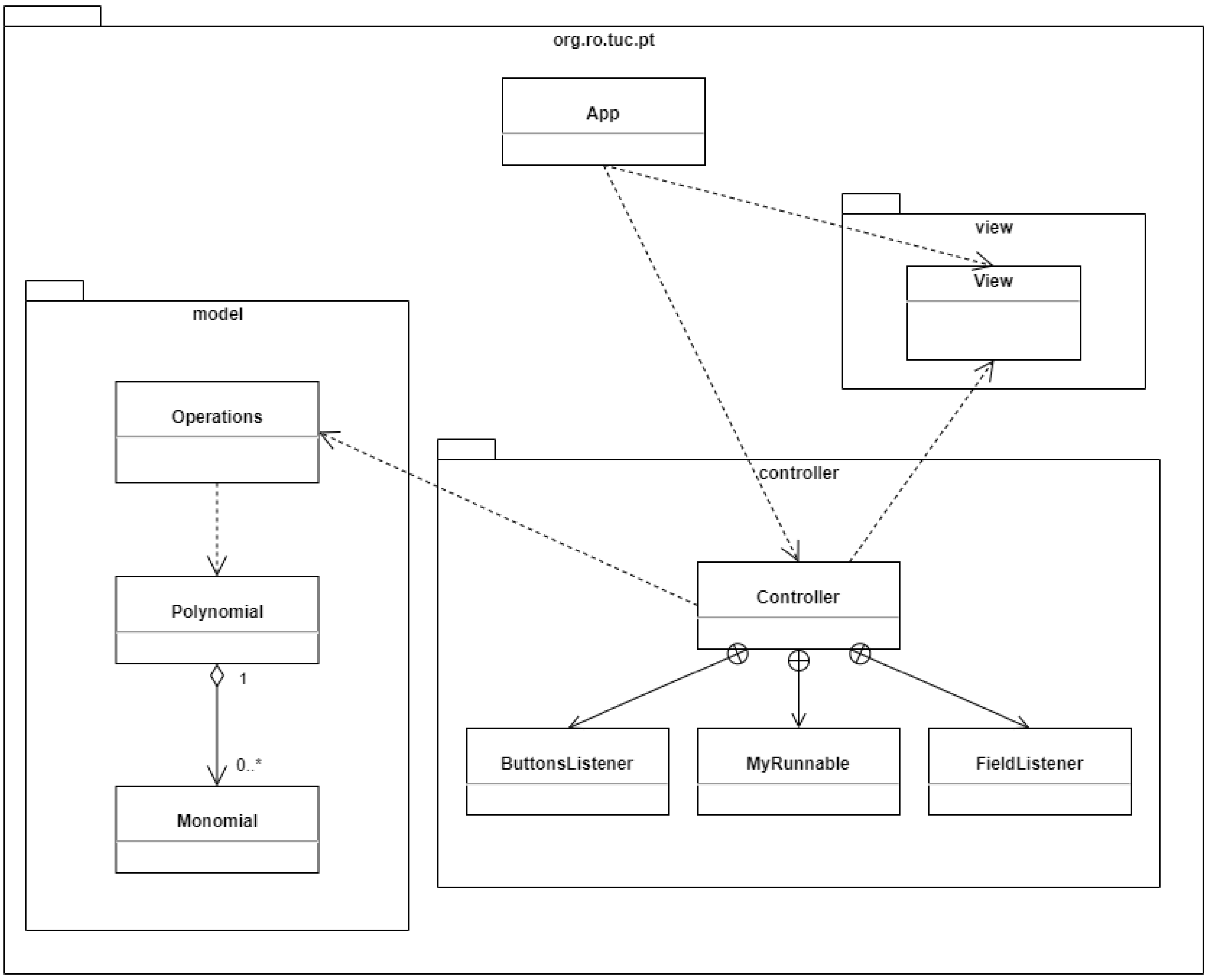
MyRunnable	
• Validates input • Requires the computations • Gives the result to be displayed	• Operations • Polynomial • ButtonsListener • View

Operations	
• Validates input • Performs computations	• Polynomial • Monomial • Controller

Polynomial	
• Organizes data • Performs operations on itself • Gives its String representation	• Monomial

Monomial	
• Performs operations on itself • Gives its String representation	• Polynomial

3.2.2 Class diagram



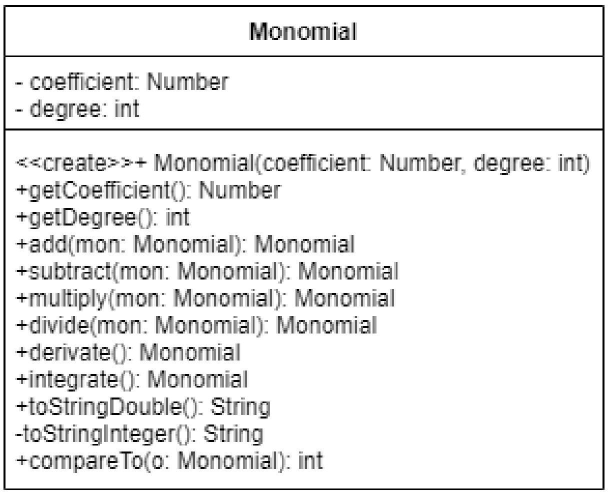
4. IMPLEMENTATION

4.1 Class descriptions

4.1.1 Class Monomial

It implements Comparable <Monomial>.

It has an integer degree field and a Number coefficient (to be able to accommodate double values in the case of division and integration operation, in case of need).



```
Monomial
- coefficient: Number
- degree: int
<<create>>+ Monomial(coefficient: Number, degree: int)
+getCoefficient(): Number
+getDegree(): int
+add(mon: Monomial): Monomial
+subtract(mon: Monomial): Monomial
+multiply(mon: Monomial): Monomial
+divide(mon: Monomial): Monomial
+derivate(): Monomial
+integrate(): Monomial
+toStringDouble(): String
-toStringInteger(): String
+compareTo(o: Monomial): int
```

The add and subtract methods resemble each other. They check if the argument is null and if the monomial and the argument have equal degrees. They return a new monomial in case of success and a null value in case of failure.

```
public Monomial add(Monomial mon)
{
    if(mon==null) return null;
    if(mon.getDegree()!= degree)
        return null;
    Monomial rez=new Monomial(coefficient.doubleValue() +
mon.getCoefficient().doubleValue(), degree);
    return rez;
}
```

The multiply and divide methods only check for null arguments.

```
public Monomial multiply(Monomial mon)
{
    if(mon==null) return null;
    Monomial rez=new Monomial(coefficient.doubleValue() *
mon.getCoefficient().doubleValue(), degree +
mon.getDegree());
    return rez;
}
```

The derivate and integrate operations take no argument. They return a new Monomial object. In case of derivation, if the degree is zero then the method returns a null.

```
public Monomial derivate()
{
    if (degree==0) return null;
    return new
Monomial(coefficient.intValue()*degree,degree-1);
}

public Monomial integrate()
{
    return new
Monomial(coefficient.doubleValue()/(degree+1),degree+1);
}
```

The toStringDouble() method calls the toStringInteger() method if the coefficient is integer. They return the Monomial in a nice String form.

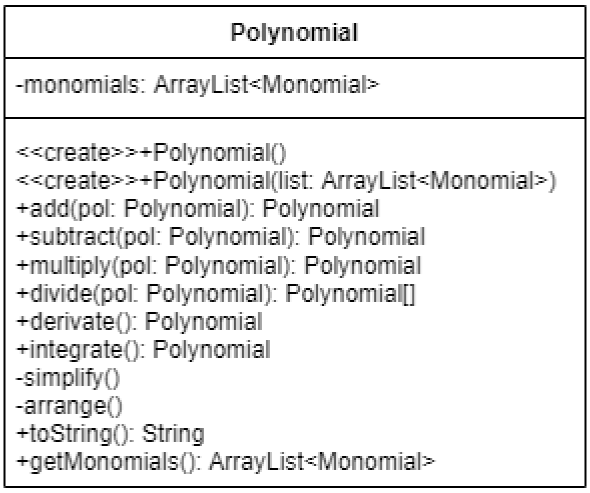
The compareTo method is used when ordering the monomials by their degrees for a nice output.

```
@Override
public int compareTo(Monomial o) {
    if(this.degree>o.getDegree()) return 1;
    if(this.degree<o.getDegree()) return -1;
    return 0;
}
```

4.1.2 Class Polynomial

A polynomial is composed of a list of monomials.
The no-parameters constructor creates an empty polynomial.

The add and subtract methods create a polynomial with a new list containing the monomials of the argument and of the current polynomial then call simplify().



```
Polynomial
-monomials: ArrayList<Monomial>
<<create>>+Polynomial()
<<create>>+Polynomial(list: ArrayList<Monomial>)
+add(pol: Polynomial): Polynomial
+subtract(pol: Polynomial): Polynomial
+multiply(pol: Polynomial): Polynomial
+divide(pol: Polynomial): Polynomial[]
+derivate(): Polynomial
+integrate(): Polynomial
-simplify()
-arrange()
+toString(): String
+getMonomials(): ArrayList<Monomial>
```

```
public Polynomial subtract(Polynomial pol)
{
    if(pol==null) return null;

    ArrayList<Monomial>monomials2= pol.getMonomials();

    ArrayList<Monomial>newList = new
ArrayList<Monomial>();
    for(Monomial mon:monomials)
        newList.add(mon);
    for(Monomial mon2: monomials2)
    {
        newList.add(new 
Monomial(mon2.getCoefficient().doubleValue()*(-
1),mon2.getDegree()));
    }

    Polynomial rez= new Polynomial(newList);
    rez.simplify();
    return rez;
}
```

```
public Polynomial multiply(Polynomial pol)
{
    if(pol==null) return null;

    ArrayList<Monomial>monomials2=
pol.getMonomials();

    ArrayList<Monomial>newList = new 
ArrayList<Monomial>();

    for(Monomial mon:monomials)
    {
        for(Monomial mon2: monomials2)
        {
            newList.add(mon.multiply(mon2));
        }
    }
    Polynomial rez = new Polynomial(newList);
    rez.simplify();
    return rez;
}
```

The derivate and integrate methods call the corresponding methods of the Monomial class on each monomial in the list and create a new polynomial.

The divide method returns an array of polynomials of length 2. On position 0 being the quotient, and on the position 1 the remainder of the division.

```
public Polynomial[] divide(Polynomial pol)
{
    Polynomial[] rezult= new Polynomial[2]; //[0] 
quotient, [1] remainder
    ArrayList<Monomial>quotient = new 
ArrayList<Monomial>();
    Polynomial deimpartit=new Polynomial(monomials);

    int 
deg1=deimpartit.getMonomials().get(0).getDegree();
    int deg2=pol.getMonomials().get(0).getDegree();
    if(deg2>deg1)
    {
        rezult[0]=new Polynomial();
        rezult[1]=this;
    }
    else{
        while(deg1>=deg2)
        {
            Monomial
mon1=deimpartit.getMonomials().get(0);
            Monomial mon2=pol.getMonomials().get(0);
            Monomial q=mon1.divide(mon2);
            quotient.add(q);
            ArrayList<Monomial>toMultiply = new
ArrayList<Monomial>();
            toMultiply.add(q);

deimpartit=deimpartit.subtract(pol.multiply(new
Polynomial(toMultiply)));
            if(deimpartit.getMonomials().size()>=1)

deg1=deimpartit.getMonomials().get(0).getDegree();
            else break;
        }
    }
    rezult[0]=new Polynomial(quotient);
    rezult[1]=deimpartit;
    return rezult;
}
```

The simplify method takes all the monomials having the same degree and adds their coefficients, then creates a new monomial with the degree and the resulting coefficient to replace them.

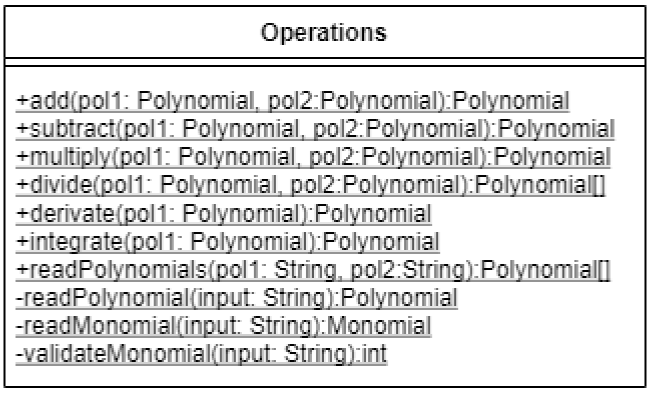
```
private void simplify()
{
    ArrayList<Monomial>newList = new
ArrayList<Monomial>();
    boolean[] degreespozitive = new boolean[10000];
    boolean[] degreesnegative = new boolean[10000];
    for(int j=0; j<10000; j++) {
        degreespozitive[j]=false;
        degreesnegative[j]=false;
    }
    int i=0;
    for (Monomial mon: monomials) {
        double coeff =
mon.getCoefficient().doubleValue();
        int deg= mon.getDegree();
        if((deg>=0 && !degreespozitive[deg]) || (deg<0 &&
!degreesnegative[-deg])){ //we haven't done this degree
yet
```

```
                for(Monomial mon2:
monomials.subList(i+1,monomials.size())){
                    if(deg == mon2.getDegree())
                        coeff+=
mon2.getCoefficient().doubleValue();
                }
                if (deg>=0) degreespozitive[deg]=true;
                else degreesnegative[-deg]=true;

                if(coeff!=0){
                    if((int)coeff==coeff)
                        newList.add(new Monomial((int)coeff,
deg));
                    else newList.add(new Monomial(coeff,
deg));}
            }
            i++;
        }
        this.monomials=newList;
        this.arrange();
}
```

The toString method prints every monomial from the list of monomials separating their sign with a space.

4.1.3 Class Operations

The class has no fields and all the methods are static. The add, subtract, multiply, divide, derivate, integrate methods call the corresponding methods from class Polynomial.



```
Operations
+add(pol1: Polynomial, pol2:Polynomial):Polynomial
+subtract(pol1: Polynomial, pol2:Polynomial):Polynomial
+multiply(pol1: Polynomial, pol2:Polynomial):Polynomial
+divide(pol1: Polynomial, pol2:Polynomial):Polynomial[]
+derivate(pol1: Polynomial):Polynomial
+integrate(pol1: Polynomial):Polynomial
+readPolynomials(pol1: String, pol2:String):Polynomial[]
-readPolynomial(input: String):Polynomial
-readMonomial(input: String):Monomial
-validateMonomial(input: String):int
```

The readPolynomials method checks for empty strings and calls the readPolynomial method for both of them. It returns an array of Polynomial having a null value if the input was incorrect.

The readPolynomial method tokenizes the input and calls the readMonomial method. It returns a new Polynomial or null in case of faulty input.

```
private static Polynomial readPolynomial(String
input)
{
    if (input==null) return null;
    Polynomial rezult;
```

```
        Monomial monomial;
        StringTokenizer tok = new
StringTokenizer(input," ");
        ArrayList<Monomial> monomials = new
ArrayList<Monomial>();
        int sign=1;
        String str;
        while(tok.hasMoreElements())
        {
            str=tok.nextToken();
            if(str.compareTo("-")==0) sign =-1;
            else if(str.compareTo("+")==0) sign =1;
                else {
                    monomial=readMonomial(str);
                    if(monomial==null)
                        return null;
                    else if(sign==1)
monomials.add(monomial);
                          else monomials.add(new
Monomial(monomial.getCoefficient().doubleValue()*
(-1),monomial.getDegree()));
                }
        }
        rezult = new Polynomial(monomials);
        return rezult;
}
```

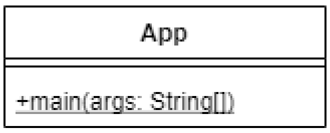
The readMonomial method calls the validateMonomial method that returns an int value representing the type of monomial it was sent (0 – just a number, 1 – number and x to power 1, 2 – number and x to a positive power, 3 – number and x to a negative power, 4 – x to power 1, 5 – x to a positive power, 6 – x to a negative power, -1 if faulty input) and then reads the value for the coefficient and the degree according to it.

```
private static Monomial readMonomial(String input)
{
    if (input==null) return null;
    int coeff, degree;
    int val=validateMonomial(input);
    if(val==-1)
        return null;
    switch (val)
    {
        case 0:
            coeff=Integer.parseInt(input);
            return new Monomial(coeff,0);
        case 1:
```

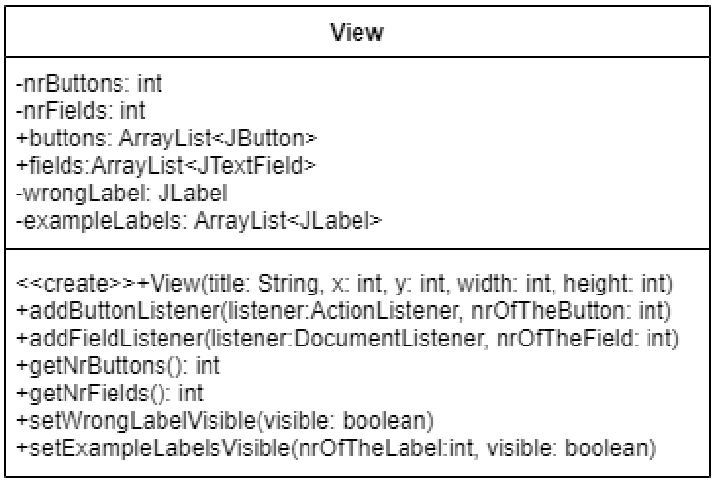
```
coeff=Integer.parseInt(input.substring(0,input.length()-
1));
                return new Monomial(coeff,1);
            case 2:
            case 3:
                int place=input.indexOf("x^");

coeff=Integer.parseInt(input.substring(0,place));

degree=Integer.parseInt(input.substring(place+2,input.len
gth()));
                return new Monomial(coeff,degree);
            case 4: return new Monomial(1,1);
            case 5:
            case 6: return new
Monomial(1,Integer.parseInt(input.substring(2,input.lengt
h())));
        }
        return null;
    }


private static int validateMonomial(String input)
{
    String[] formats = new String[7];
    formats[0] = "[0-9]{1,4}"; //just a number
    formats[1] = "[0-9]{1,4}x"; // number + x to power 1
    formats[2] = "[0-9]{1,4}x\\^[0-9]{1,4}"; //number + x
to a power
    formats[3] = "[0-9]{1,4}x\\^-[0-9]{1,4}"; //number +
x to a negative power
    formats[4] = "x"; //x to power 1
    formats[5] = "x\\^[0-9]{1,4}"; //x to a power
    formats[6] = "x\\^-[0-9]{1,4}"; //x to a negative
power

    Pattern[] patterns = new Pattern[7];
    Matcher[] matchers = new Matcher[7];
    for(int i=0; i<7;i++)
    {
        patterns[i] = Pattern.compile(formats[i],
Pattern.CASE_INSENSITIVE);
        matchers[i]=patterns[i].matcher(input);
        if(matchers[i].matches())
            return i;
    }
    return -1; //not found
}
```

4.1.4 Class App

The App class contains the main method which Creates the View and the Controller objects.



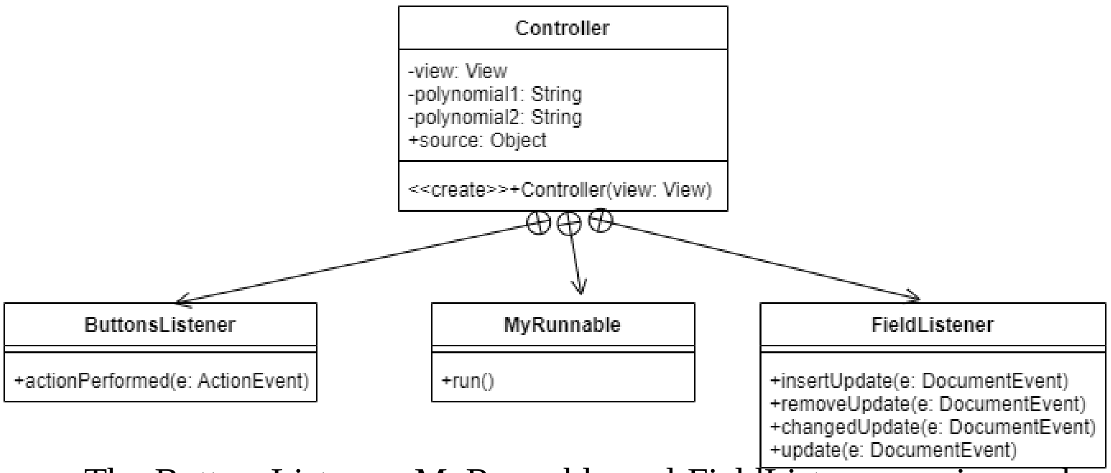
4.1.5 Class View


View
-nrButtons: int
-nrFields: int
+buttons: ArrayList<JButton>
+fields:ArrayList<JTextField>
-wrongLabel: JLabel
-exampleLabels: ArrayList<JLabel>
<<create>>+View(title: String, x: int, y: int, width: int, height: int)
+addButtonListener(listener:ActionListener, nrOfTheButton: int)
+addFieldListener(listener:DocumentListener, nrOfTheField: int)
+getNrButtons(): int
+getNrFields(): int
+setWrongLabelVisible(visible: boolean)
+setExampleLabelsVisible(nrOfTheLabel:int, visible: boolean)

The class extends JFrame. The ArrayList buttons and fields are public because they are needed in Controller for identifying the source of the action that was performed.

The constructor creates all the components needed in the view. The addButtonListener and addFieldListener methods are used for assigning to each button and field a listener. The setWrongLabelVisible and setExampleLabelsVisible methods set the respective label visible or not.

4.1.6 Class Controller


Controller
-view: View
-polynomial1: String
-polynomial2: String
+source: Object
<<create>>+Controller(view: View)
ButtonsListener
+actionPerformed(e: ActionEvent)
MyRunnable
+run()
FieldListener
+insertUpdate(e: DocumentEvent)
+removeUpdate(e: DocumentEvent)
+changedUpdate(e: DocumentEvent)
+update(e: DocumentEvent)

The ButtonsListener, MyRunnable and FieldListener are inner classes of Controller class.

The constructor assigns ButtonsListener objects to the view's buttons, and FieldListener to the view's fields.

```
public Controller(View view)
{
    this.view=view;

    int nrButtons=view.getNrButtons();
    for(int j=0;j<nrButtons;j++)
        view.addButtonListener(new ButtonsListener(),j);

    int nrFields=view.getNrFields() -1;
    for(int k=0;k<nrFields;k++)
        view.addFieldListener(new FieldListener(),k);
}
```

4.1.7 Class ButtonsListener

The class implements the interface ActionListener. The actionPerformed method sets the Controller's source and starts the method run() from the class MyRunnable in a new thread.

```
@Override
public void actionPerformed(ActionEvent e) {
    Controller.this.source=e.getSource();
    view.fields.get(2).setText(null); //the result field

    Thread thread = new Thread(new MyRunnable()); //to
perform the computations needed in another thread
    thread.start();
}
```

4.1.8 Class MyRunnable

The class implements the interface Runnable. The method run() calls the Operation's methods to compute the results and display it on the view, or display the error message in case of incorrect input.

```
@Override
public void run() {
    Polynomial[] reading =
Operations.readPolynomials(polynomial1,polynomial2);
    view.setWrongLabelVisible(false);
    if(source==view.buttons.get(0)){ //add
        if(reading[0]==null || reading[1]==null)
            view.setWrongLabelVisible(true);
        else
view.fields.get(2).setText(Operations.add(reading[0],read
ing[1]).toString());
    } else if(source==view.buttons.get(1)){ //subtract
                if(reading[0]==null || reading[1]==null)
                    view.setWrongLabelVisible(true);
                else
view.fields.get(2).setText(Operations.subtract(reading[0]
,reading[1]).toString());
```

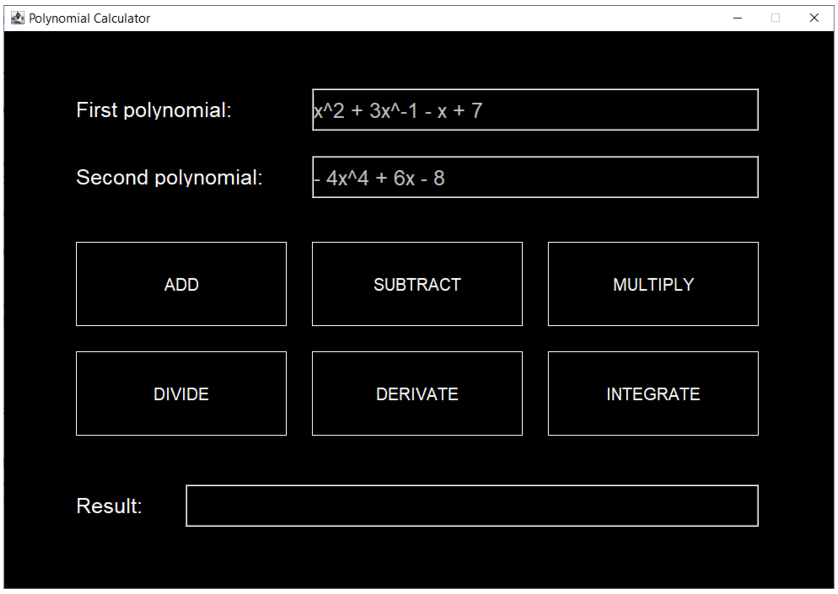
```
            } else if(source==view.buttons.get(2)){
//multiply
                        if(reading[0]==null ||
reading[1]==null)

view.setWrongLabelVisible(true);
                        else
view.fields.get(2).setText(Operations.multiply(reading[0]
,reading[1]).toString());
                    } else
if(source==view.buttons.get(3)){ //divide
                                if(reading[0]==null ||
reading[1]==null ||
reading[1].toString().compareTo("0")==0)

view.setWrongLabelVisible(true);
                                else{
                                    Polynomial[] division
= Operations.divide(reading[0],reading[1]);

view.fields.get(2).setText("quotient:
"+division[0].toString()+", remainder:
"+division[1].toString());
                                }
                            } else
if(source==view.buttons.get(4)){ //derivate

if(reading[0]==null)

view.setWrongLabelVisible(true);
                                        else
view.fields.get(2).setText(Operations.derivate(reading[0]
).toString());
                                    } else
if(source==view.buttons.get(5)){ //integrate

if(reading[0]==null)

view.setWrongLabelVisible(true);
                                                else
view.fields.get(2).setText(Operations.integrate(reading[0
]).toString());
                                            }
}
```

4.1.9 Class FieldListener

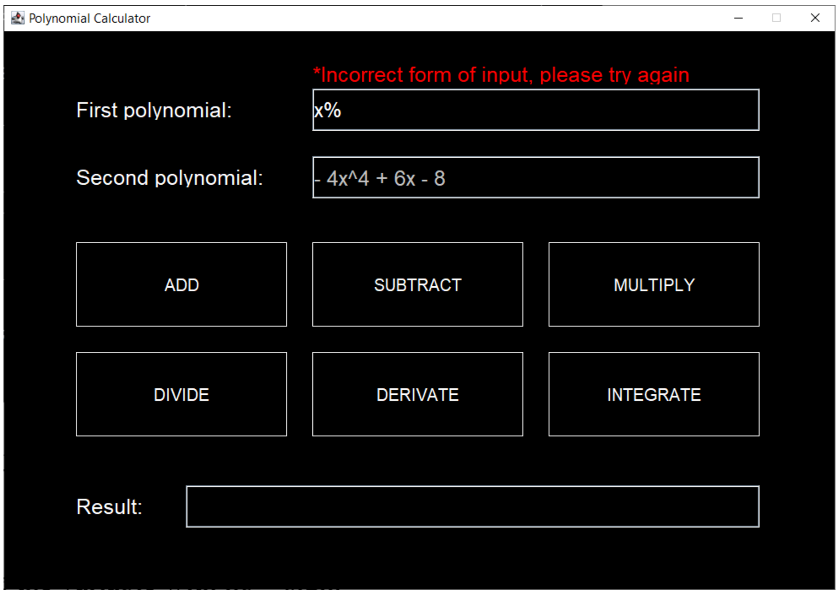
The class implements the interface DocumentListener. The update method reads in the Controller's polynomial1 and polynomial2 Strings the corresponding input from the user and shows (or not) the example labels from the view.

```
public void update(DocumentEvent e)
{
    Object sourceDocument = e.getDocument();
    if(sourceDocument==view.fields.get(0).getDocument())
//polynomial one
    {
        Controller.this.polynomial1 =
view.fields.get(0).getText();
        if(polynomial1==null || polynomial1.isEmpty())
            view.setExampleLabelsVisible(0,true);
        else view.setExampleLabelsVisible(0,false);
        return;
    }
    if(sourceDocument==view.fields.get(1).getDocument())
//polynomial two
    {
        Controller.this.polynomial2 =
view.fields.get(1).getText();
        if(polynomial2==null || polynomial2.isEmpty())
            view.setExampleLabelsVisible(1,true);
        else view.setExampleLabelsVisible(1,false);
        return;
    }
}
```

4.2 GUI description



The GUI has 3 fields, the first 2 are used for entering the inputs and the third for displaying the result. There are 2 examples illustrating correct inputs. The + and – signs must be at a space distance from the following monomial. The coefficient is directly before the variable with no * sign.



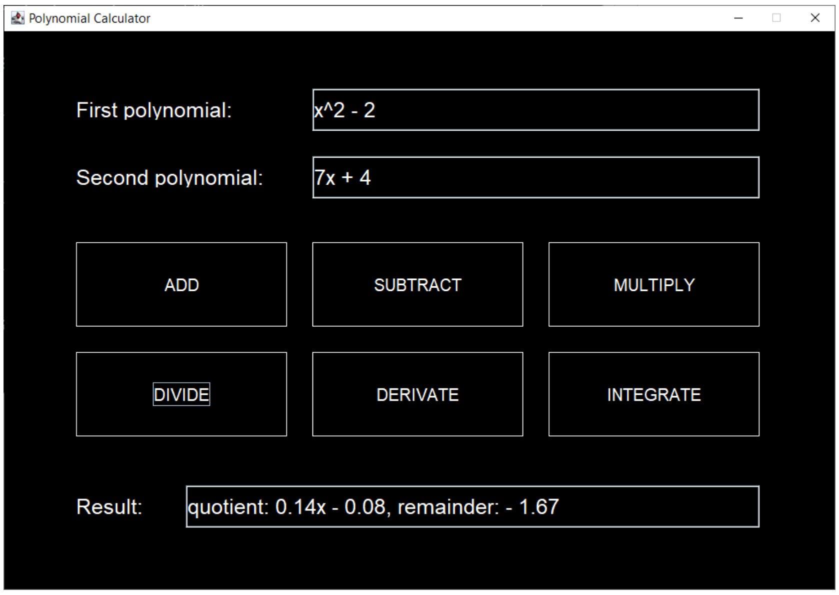
*Incorrect form of input, please try again

First polynomial: x%

Second polynomial: - 4x^4 + 6x - 8

Result:

If an incorrect input is given, when a button is pressed, an error message is displayed.



First polynomial: x^2 - 2

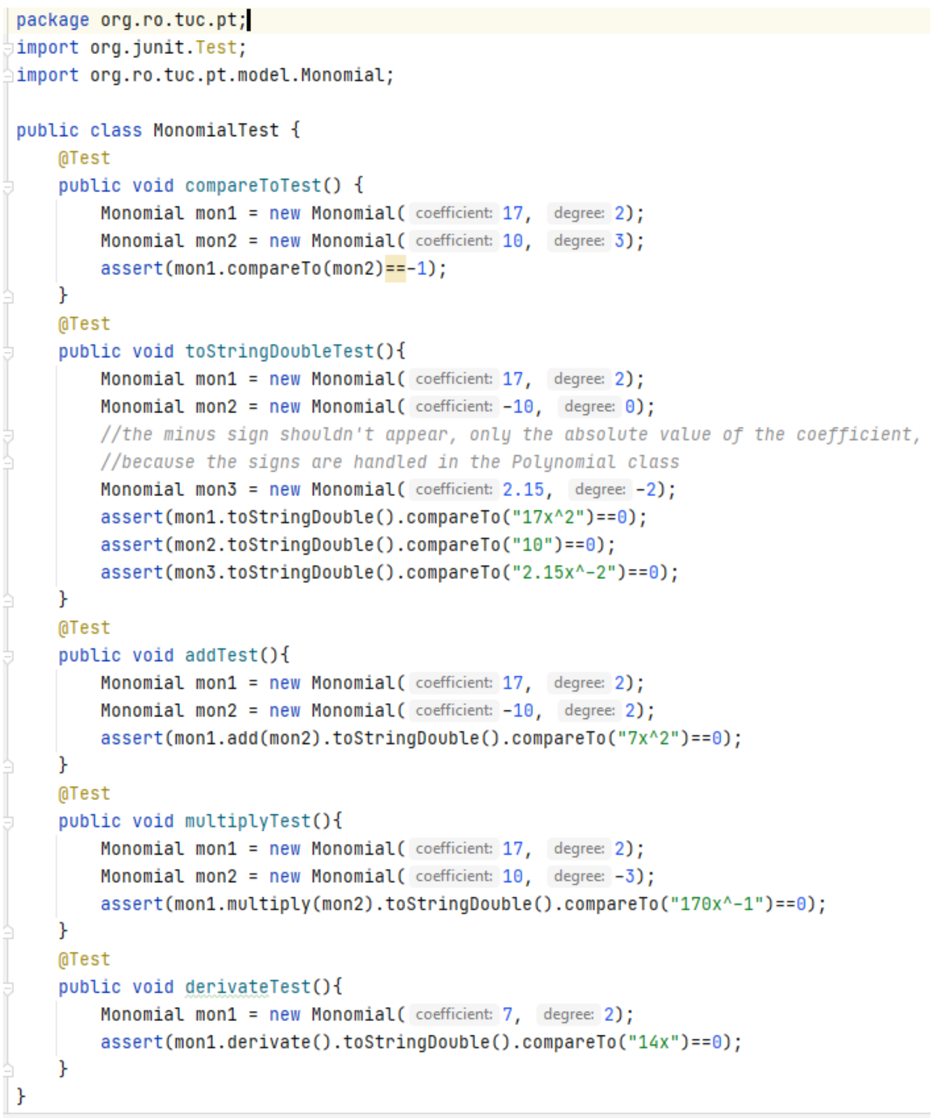
Second polynomial: 7x + 4

Result: quotient: 0.14x - 0.08, remainder: - 1.67

If the input is correct, after a button is pressed, the result is displayed.

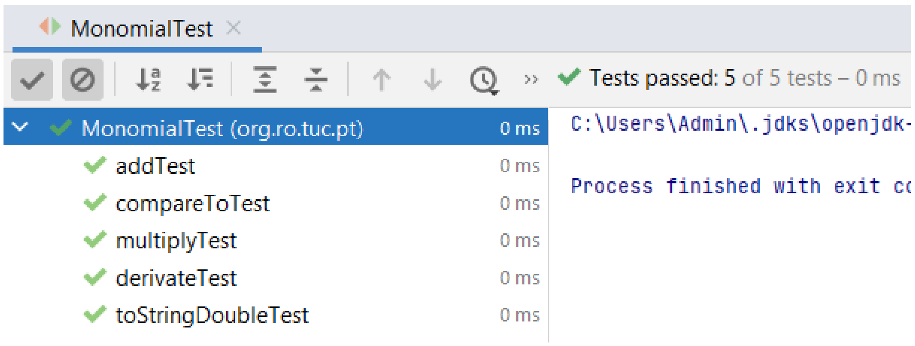
5. RESULTS

5.1 Monomial test



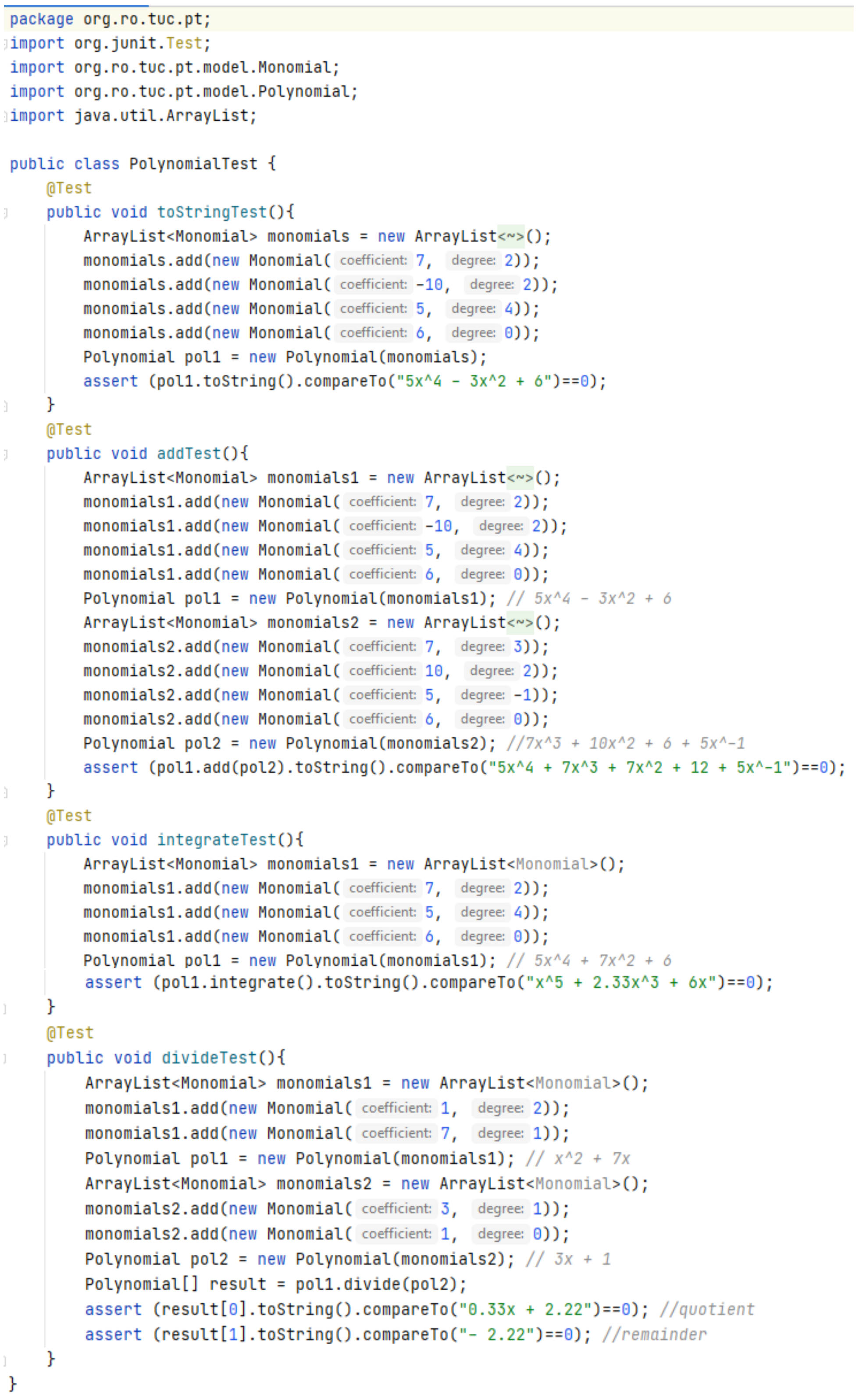
```
package org.ro.tuc.pt;
import org.junit.Test;
import org.ro.tuc.pt.model.Monomial;

public class MonomialTest {
    @Test
    public void compareToTest() {
        Monomial mon1 = new Monomial( coefficient: 17,  degree: 2);
        Monomial mon2 = new Monomial( coefficient: 10,  degree: 3);
        assert(mon1.compareTo(mon2)==-1);
    }
    @Test
    public void toStringDoubleTest(){
        Monomial mon1 = new Monomial( coefficient: 17,  degree: 2);
        Monomial mon2 = new Monomial( coefficient: -10,  degree: 0);
        //the minus sign shouldn't appear, only the absolute value of the coefficient,
        //because the signs are handled in the Polynomial class
        Monomial mon3 = new Monomial( coefficient: 2.15,  degree: -2);
        assert(mon1.toStringDouble().compareTo("17x^2")==0);
        assert(mon2.toStringDouble().compareTo("10")==0);
        assert(mon3.toStringDouble().compareTo("2.15x^-2")==0);
    }
    @Test
    public void addTest(){
        Monomial mon1 = new Monomial( coefficient: 17,  degree: 2);
        Monomial mon2 = new Monomial( coefficient: -10,  degree: 2);
        assert(mon1.add(mon2).toStringDouble().compareTo("7x^2")==0);
    }
    @Test
    public void multiplyTest(){
        Monomial mon1 = new Monomial( coefficient: 17,  degree: 2);
        Monomial mon2 = new Monomial( coefficient: 10,  degree: -3);
        assert(mon1.multiply(mon2).toStringDouble().compareTo("170x^-1")==0);
    }
    @Test
    public void derivateTest(){
        Monomial mon1 = new Monomial( coefficient: 7,  degree: 2);
        assert(mon1.derivate().toStringDouble().compareTo("14x")==0);
    }
}
```

The class MonomialTest was used for unit testing the Monomial class. The methods toStringDouble, add, multiply and derivate were tested using manually introduced data. The toStringDouble method was verified and then it was used for testing the correctness of the other methods.



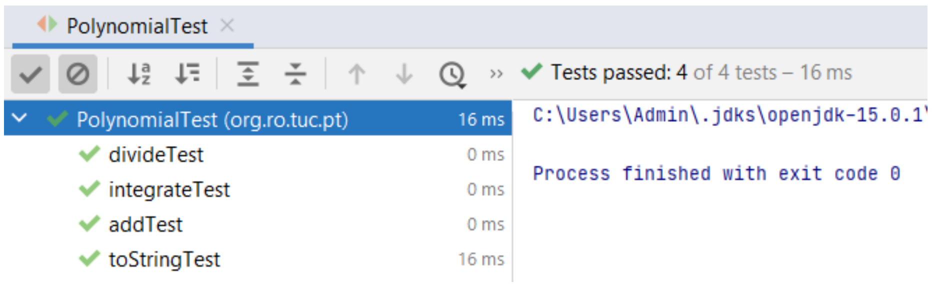
```
MonomialTest
✓ Tests passed: 5 of 5 tests – 0 ms
✓ MonomialTest (org.ro.tuc.pt)   0 ms    C:\Users\Admin\.jdks\openjdk-
    ✓ addTest                    0 ms
    ✓ compareToTest              0 ms    Process finished with exit co
    ✓ multiplyTest               0 ms
    ✓ derivateTest               0 ms
    ✓ toStringDoubleTest         0 ms
```

5.2 Polynomial test



```java
package org.ro.tuc.pt;
import org.junit.Test;
import org.ro.tuc.pt.model.Monomial;
import org.ro.tuc.pt.model.Polynomial;
import java.util.ArrayList;

public class PolynomialTest {
    @Test
    public void toStringTest(){
        ArrayList<Monomial> monomials = new ArrayList<~>();
        monomials.add(new Monomial( coefficient: 7,  degree: 2));
        monomials.add(new Monomial( coefficient: -10,  degree: 2));
        monomials.add(new Monomial( coefficient: 5,  degree: 4));
        monomials.add(new Monomial( coefficient: 6,  degree: 0));
        Polynomial pol1 = new Polynomial(monomials);
        assert (pol1.toString().compareTo("5x^4 - 3x^2 + 6")==0);
    }
    @Test
    public void addTest(){
        ArrayList<Monomial> monomials1 = new ArrayList<~>();
        monomials1.add(new Monomial( coefficient: 7,  degree: 2));
        monomials1.add(new Monomial( coefficient: -10,  degree: 2));
        monomials1.add(new Monomial( coefficient: 5,  degree: 4));
        monomials1.add(new Monomial( coefficient: 6,  degree: 0));
        Polynomial pol1 = new Polynomial(monomials1); // 5x^4 - 3x^2 + 6
        ArrayList<Monomial> monomials2 = new ArrayList<~>();
        monomials2.add(new Monomial( coefficient: 7,  degree: 3));
        monomials2.add(new Monomial( coefficient: 10,  degree: 2));
        monomials2.add(new Monomial( coefficient: 5,  degree: -1));
        monomials2.add(new Monomial( coefficient: 6,  degree: 0));
        Polynomial pol2 = new Polynomial(monomials2); //7x^3 + 10x^2 + 6 + 5x^-1
        assert (pol1.add(pol2).toString().compareTo("5x^4 + 7x^3 + 7x^2 + 12 + 5x^-1")==0);
    }
    @Test
    public void integrateTest(){
        ArrayList<Monomial> monomials1 = new ArrayList<Monomial>();
        monomials1.add(new Monomial( coefficient: 7,  degree: 2));
        monomials1.add(new Monomial( coefficient: 5,  degree: 4));
        monomials1.add(new Monomial( coefficient: 6,  degree: 0));
        Polynomial pol1 = new Polynomial(monomials1); // 5x^4 + 7x^2 + 6
        assert (pol1.integrate().toString().compareTo("x^5 + 2.33x^3 + 6x")==0);
    }
    @Test
    public void divideTest(){
        ArrayList<Monomial> monomials1 = new ArrayList<Monomial>();
        monomials1.add(new Monomial( coefficient: 1,  degree: 2));
        monomials1.add(new Monomial( coefficient: 7,  degree: 1));
        Polynomial pol1 = new Polynomial(monomials1); // x^2 + 7x
        ArrayList<Monomial> monomials2 = new ArrayList<Monomial>();
        monomials2.add(new Monomial( coefficient: 3,  degree: 1));
        monomials2.add(new Monomial( coefficient: 1,  degree: 0));
        Polynomial pol2 = new Polynomial(monomials2); // 3x + 1
        Polynomial[] result = pol1.divide(pol2);
        assert (result[0].toString().compareTo("0.33x + 2.22")==0); //quotient
        assert (result[1].toString().compareTo("- 2.22")==0); //remainder
    }
}
```

The class PolynomialTest was used for unit testing the Polynomial class. The methods toString, add,integrate and divide were tested using manually introduced data. The toString method was verified and then it was used for testing the correctness of the other methods.



PolynomialTest

Tests passed: 4 of 4 tests – 16 ms

```
PolynomialTest (org.ro.tuc.pt)   16 ms
    divideTest                    0 ms
    integrateTest                 0 ms
    addTest                       0 ms
    toStringTest                 16 ms
```

```
C:\Users\Admin\.jdks\openjdk-15.0.1

Process finished with exit code 0
```

6. CONCLUSIONS

6.1 What I have learned

- How to use the Model-View-Controller architectural pattern.
- How to divide a problem into subsystems/packages
- How to use JUnit for testing
- How to represent the class-inner class relationship in a class diagram
- How to represent a static method in a class diagram
- How to use git
- How to use pattern matchings
- How to use the Number class
- What are functional and non-functional requirements
- How to make use cases
- How to structure a project documentation
- That the project documentation takes a lot of time to complete

6.2 Future development

- The application could accept polynomials of two or more variables and the integration and derivation operations could take as an argument, the variable to which the operation is performed.
- The application could accept polynomials with non-integer coefficients (even complex).
- There could be an operation to calculate the value of the polynomial in a point, or the value of the integral on an interval.
- The application could calculate the lowest common denominator and the greatest common divisor of two polynomials.
- The polynomials could be taken as functions and the points of extrema could be calculated.

7. BIBLIOGRAPHY

- https://www.uml-diagrams.org/nested-classifier.html
- https://app.diagrams.net/
- https://echeung.me/crcmaker/
- https://notepad2.blogspot.com/2012/04/how-to-indicate-static-method-in-uml.html?m=1&fbclid=IwAR3zgihrTUmcKx3EiV8khKnpKe3W0uVkfWUThkyP1AR6WAHHGUoBaAgufrM
- https://docs.oracle.com/javase/7/docs/api/java/util/StringTokenizer.html?fbclid=IwAR24FGLHJtCc_lQURKjutPfNGsqpsawDf5lpYYpm4ejGg0eIm6q2YU7JhWQ
- https://stackoverflow.com/questions/16706716/using-two-values-for-one-switch-case-statement?fbclid=IwAR1wnidu7uwespz37cPaVMa3PoK186GY9QjoUMdk6M-mCUPLAlAhPbDXPuU
- https://stackoverflow.com/questions/32492245/java-regex-optional-matching/32492445?fbclid=IwAR36E31OeLnZNzeiCooGf_wCDXTVOLRoUjrfru0E2UjWhByMyTfMHOewtcY
- https://stackoverflow.com/questions/26054909/regex-with-variable-length?fbclid=IwAR3STXeWtm1oqtP3znRco8czrd1kBTBZyVCFXj-wocAZk79MPMUuNr0DukI
- https://stackoverflow.com/questions/965690/java-optional-parameters?fbclid=IwAR1SHEyKgEdmGR0mGROJwSJWSB6Y6LVcxvkod7KnGc__W4gWJs_cEr7hwuA
- https://www.tutorialspoint.com/how-to-get-first-and-last-elements-from-arraylist-in-java?fbclid=IwAR3zgihrTUmcKx3EiV8khKnpKe3W0uVkfWUThkyP1AR6WAHHGUoBaAgufrM
- https://howtodoinjava.com/java/sort/collections-sort/?fbclid=IwAR3k03XF-SIBcy9XgJdWjlFPzuvePoUTcJwcN_zkW-ELOVodQzLSUCVeCgM
- https://stackoverflow.com/questions/16802618/nested-foreach-statement-in-java?fbclid=IwAR0CmGVb6ezvKM5IxmT9CZSNGY1l1-TU43mTgnd8SttNU10p60TknB9-XyE
- https://stackoverflow.com/questions/1196586/calling-remove-in-foreach-loop-in-java?fbclid=IwAR1Yb0BhmIxu8HSHWPFqNiP-GpbfMoREM0dVZBuqJt_9HpUN0o58uzoBAuw