Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

2nd Semester 2018-2019

Programming Techniques Homework 1

Student: Elekes Lukacs-Roland

Group: 30424

Supervising teacher: Viorica Chifu

Table of Contents

1. Objectives ... 3
2. Problem Analysis, Modelling, Scenarios, Use Cases ... 4
 2.1 Problem Analysis ... 4
 2.2 Modelling ... 4
 2.3 Scenarios, use-cases ... 5
3. Design ... 8
 3.1 Class Design and UML Diagram ... 8
 3.2 Packages and Relationships ... 9
 3.3 Algorithms ... 9
 3.4 Graphical User Interface ... 10
4. Implementation ... 11
 4.1 Class Monom ... 11
 4.2 Class Polynomial ... 12
 4.3 Class PolynomialView ... 14
 4.4 Class PolynomialController ... 14
 4.5 Class Main ... 16
5. Usage and Testing ... 16
6. Conclusions ... 17
7. Bibliography ... 18

1. Objectives

The main objective is to propose, design and implement a system for polynomial processing. Consider the polynomials of one variable and integer coefficients.

In order to achieve the main objective, we have to take in consideration the following secondary objectives, implementation of:

- Addition of polynomials (chapters: 3.3, 4.2)
- Subtraction of polynomials (chapters: 3.3, 4.2)
- Multiplication of polynomials (chapters: 3.3, 4.2)
- Division of polynomials (chapters: 3.3, 4.2)
- Derivation of polynomials (chapters: 3.3, 4.2)
- Integration of polynomials (chapters: 3.3, 4.2)
- Graphical user interface (chapter: 3.4, 4.3)

2.Problem Analysis, Modelling, Scenarios, Use Cases

2.1 Problem Analysis

A system for processing polynomials may be useful in almost any environment that uses mathematics. Polynomials and operations are frequently used in engineering processes, but such a system can be used by non-professionals too, for example by students to verify solutions resulted by operations on polynomials.

The fact that this system can be used in many different domains is because it provides the basic and important operations on polynomials. The user can use this application for operations like addition, subtraction, multiplication or division of two polynomials and derivation or integration of a polynomial. The polynomials are considered of one variable and integer coefficients.

A graphical user interface is provided in order to make the usage simpler than the usage of a console-based application.

2.2 Modelling

For a system for processing polynomials the main objectives in terms of modelling is the modelling of a polynomial. A polynomial consists of a series of monomials, this means that a good model for a monomial helps modelling the polynomial as an entity.

Monomials have two important attributes: coefficient and exponent. Also, on the level of monomials operations for them are implemented, which makes easier the design of operations of polynomials. A polynomial means a list of monomials, and any action on polynomials can be designed by using the monomials’ operations.

2.3 Scenarios, use-cases

The user introduces one or two polynomials (notation: P and Q); he chooses one of the operations and the result is printed for the user. If the operation is derivation or integration, then the first polynomial (P) is taken as the input for the operation.

It is assumed, that the user introduces the polynomials in a correct format and in a simplified form (if there are two terms with the same exponent, they are summed). The correct format for introducing data is: coefficient x^ exponent for every term, even if the coefficient is 1 and the exponent is 1 or 0 (terms with coefficients 0 do not need to be introduced). If nothing is introduced, it is considered the polynomial 0.

An example would be:

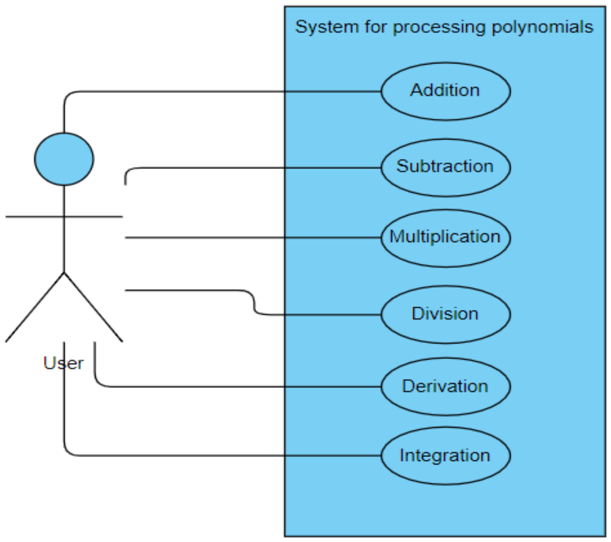
$P = 2x^2 + 3x + 2$; $Q = 3x^2 + x + 5$; $P + Q = 5x^2 + 4x + 7$;

The inputs are the following:

- For P: 2x^2+3x^1+2x^0
- For Q: 3x^2+1x^1+5x^0

After inputs an operation is selected, in our example by pressing the button "+" and a result is displayed in similar format.

The use-case diagram is the following:



Use case title: Addition

Actor: User

Main success scenario:

1. User introduces the polynomials
2. User chooses addition by pressing the button “+”
3. System takes information from input
4. The polynomials are added together
5. System displays the result

Use case title: Subtraction

Actor: User

Main success scenario

1. User introduces the polynomials
2. User chooses subtraction by pressing the button “-”
3. System takes information from input
4. The polynomials are subtracted
5. System displays the result

Use case title: Multiplication

Actor: User

Main success scenario

1. User introduces the polynomials
2. User chooses multiplication by pressing the button “*”
3. System takes information from input
4. The polynomials are multiplied
5. System displays the result

Use case title: Division

Actor: User

Main success scenario

1. User introduces the polynomials
2. User chooses division by pressing the button “/”

3. System takes information from input
4. P/Q division is performed
5. System displays the quotient and remainder

Use case title: Division

Actor: User

Alternative sequence:

Second input is 0:

1) Instead of result, system tells that division by 0 is not possible
2) The scenario returns to step 1

Use case title: Derivation

Actor: User

Main success scenario

1. User introduces the polynomial
2. User chooses Derivation by pressing the button “Derivate”
3. System takes information from the first input
4. First polynomial is differentiated
5. System displays the result

Use case title: Integration

Actor: User

Main success scenario

1. User introduces the polynomial
2. User chooses integration by pressing the button “Integrate”
3. System takes information from the first input
4. First polynomial is integrated
5. System displays the result

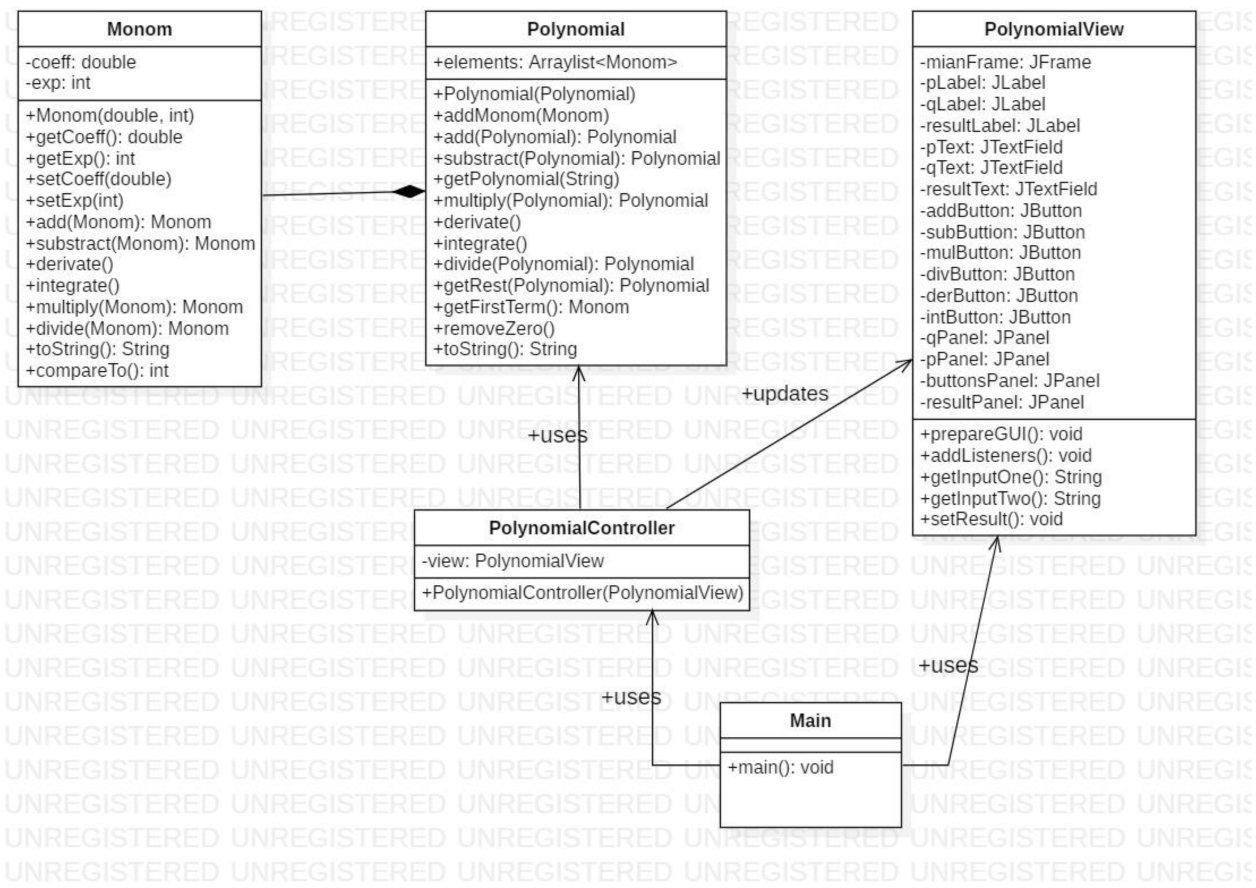
3. Design

3.1 Class Design and UML Diagram

The design of the assignment is based on the **M**odel **V**iew **C**ontroller architectural pattern, because the interaction between the user and the system is done through a **G**raphical **U**ser **I**nterface. From this pattern, the Model is the one in which we are mostly interested.

I chose to represent a polynomial as a series of monomials, thus two important classes stand at the base of the Model. There is a class called Monom that represents a monomial having as attributes a coefficient and exponent; and implements operations on the level of monomials.

The other class is named Polynomial and represents a polynomial having as attribute a list of monomials; and implements different operations on polynomials.



UML Diagram for the system for processing polynomials.

Classes like PolynomialView, PolynomialController and Main also appear, the first two being part of the MVC pattern and the Main class containing the main function for starting the application.

The class PolynomialView is responsible for the graphical user interface providing text fields (for inputs) and buttons for interacting with the system. Using a graphical user interface increases the productivity and it makes easier to handle the application than a console based application.

The class PolynomialController is the one that connects the elements from model and view. It takes data from the user (through the view, GUI) and controls the data flow to the model, then updates the view.

3.2 Packages and Relationships

According to the MVC pattern the project is structured on three packages: polinom(model), view and controller.

The package polinom contains all the classes that are part of the model (Monom and Polynomial).

The package controller contains the class PolynomialController forming the controller part of the MVC pattern and I have also placed here the Main class.

The package view contains the class PolynomialView responsible for the design of the graphical user interface presented earlier.

The relationship between classes Monom and Polynomial can be defined as composition, a variation of simple aggregation in which the child class exists only if an instance of the parent exists, since Polynomial objects are constructed from a series of Monom objects and the Monom class as standalone entity is not present in our context, even though monomials have strong meanings in other contexts for example as a polynomial with only one term (in our system these polynomials are considered as polynomial objects and not as monomials).

Furthermore, there can be observed other relationships, mainly association type relationships, like the class from controller updates the class from view and uses the Polynomial class from model.

The class Main uses both controller and view classes.

3.3 Algorithms

Some algorithms are used in the design of the model, some of them are trivial, like addition, subtraction, multiplication or division, they appear on the level of monomials and they are easy to implement since they are basic operations.

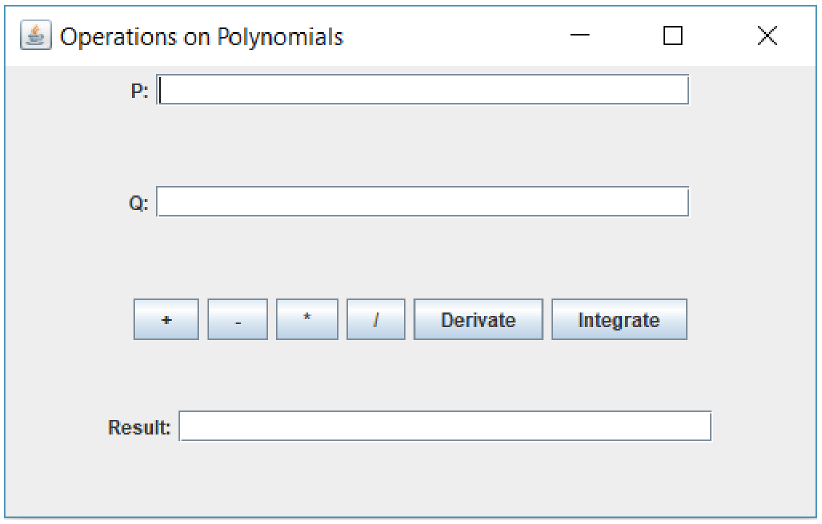
The algorithms used for the operations of polynomials are more complicated, but most of them can declared as simple ones. We can enumerate addition and subtraction, they are simple since the basic operations (already implemented for monomials) are used term-wisely. Also differentiation and integration algorithms are based on term-wisely applying the operations from monomials.

A bit more challenging is the multiplication of two polynomials, but it becomes not more complicated then multiplying every term with every term from the other polynomial, after that only the sum of terms having similar exponents has to be done.

The division of polynomials was the most interesting in terms of complexity and implementation. The algorithm is based on high school methods.

3.4 Graphical User Interface

The graphical user interface provides an easy handling of the system for the user. It has some labels (for information), text fields (for introducing data and output of the result) and buttons.

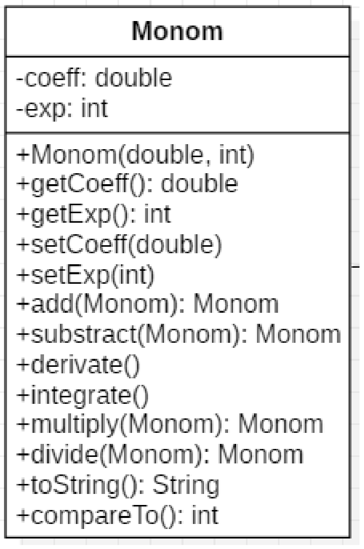


Graphical User Interface for the system for processing polynomials.

The labels on the buttons are representative symbols for the operations, after the data is introduced to the text fields, the user can execute an operation by pressing the corresponding button and the result will appear in the text field labeled as “Result:”.

4. Implementation

4.1 Class Monom



The Monom class represents a monomial and implements operations on monomials.

The field of the class are the following:

- private double coeff - The coefficient of the monomial.
- private double exp - The exponent of the monomial.

There are two constructors, one of them having two parameters for setting the attributes of the new object; and there is a constructor with no parameters which sets both attributes to 0.

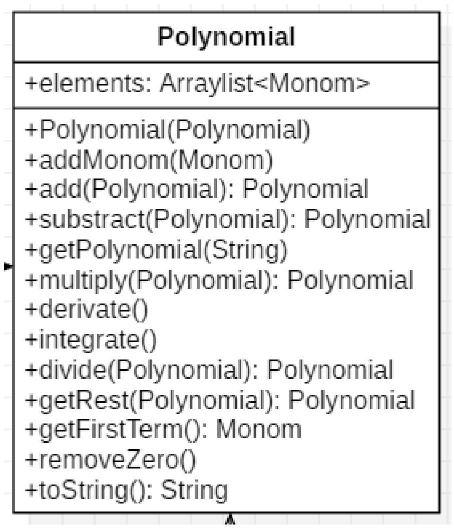
The important methods are the ones that implement operations where values are changed according to the basic operations. Some examples are shown

```
public Monom add(Monom m) {
        Monom result = new Monom();
        if (this.exp == m.exp) {
            result.coeff = this.coeff + m.coeff;
            result.exp = this.exp;
            return result;
        }
        else {
            return null;
        }

    }


public void derivate() {
        if (this.exp == 0) {
            this.coeff = 0;
        }
        else {
            this.coeff *= this.exp;
            this.exp--;
        }
    }

public Monom divide (Monom m) {
        Monom result = new Monom();
```

```
                result.setCoeff(this.coeff / m.coeff);
                result.setExp(this.exp - m.exp);
                return result;
        }
```

There exist an overridden method toString() which returns a String representing the monomial having the following form: sign + coefficient + “x^” + exponent. Also compareTo() method from the interface Comparable is implemented.

4.2 Class Polynomial


Polynomial
+elements: Arraylist<Monom>
+Polynomial(Polynomial)
+addMonom(Monom)
+add(Polynomial): Polynomial
+substract(Polynomial): Polynomial
+getPolynomial(String)
+multiply(Polynomial): Polynomial
+derivate()
+integrate()
+divide(Polynomial): Polynomial
+getRest(Polynomial): Polynomial
+getFirstTerm(): Monom
+removeZero()
+toString(): String

The Polynomial class represents a polynomial and implements operations on polynomials. The only attribute of this class is a list of type ArrayList containing Monom objects.

It provides methods that implement operations on polynomials and there are some other important and helpful methods. Some methods are presented:

❖ **public Polynomial add(Polynomial p);** - Addition of polynomials

- The polynomial **this** is copied in the **result** polynomial, then for every element of **p** is checked if a term with similar exponent exists in **result**, if yes, they are added together. If there is no match, a flag remains unchanged and that marks that the current element of **p** is simply added to the series of monomials in result with the help of the method **public void addMonom();**
- The **result** is sorted in descending order regarding the exponents and may appear monomials with coefficient 0, they are removed.

```
public Polynomial add(Polynomial p) {
                Polynomial result = new Polynomial(this);
                boolean changedCoeff = false;

                for (Monom m1 : p.elements) {
                        changedCoeff = false;
                        for(Monom m2 : result.elements) {
                                if (m1.getExp() == m2.getExp()) {
                                        int index = result.elements.indexOf(m2);
                                        m2 = m2.add(m1);
                                        result.elements.set(index, m2);
                                        changedCoeff = true;
                                        break;
                                }
                        }
                        if (!changedCoeff) {
                                result.addMonom(m1);
                        }
                }
```

```
		Collections.sort(result.elements, Collections.reverseOrder());

		result.removeZero();

		return result;
		}
```

- The **subtraction** operation is similar to this algorithm, a change appears when a match is found, the monomials are subtracted.
- **public Polynomial multiply(Polynomial p); -** Multiplicaion of polynomials.
    - Every term of **this** is multiplied by every term of **p** and they are introduced in a temporary polynomial. After this point similarly to the part where we check if there are elements with same exponent the monomials are added to the final **result**.
    - The following code shows only the part where polynomials are multiplied:

```
public Polynomial multiply(Polynomial p) {
		Polynomial temp = new Polynomial();
		for(Monom m1 : this.elements) {
			for(Monom m2 : p.elements) {
				temp.addMonom(m1.multiply(m2));
			}
		}

		Polynomial result = new Polynomial();
			boolean changedCoeff = false;
```

- **public Polynomial divide(Polynomial p);** - Division of polynomials
    - The method is based on the algorithm of dividing polynomials and returns the quotient.
    - There is a method called **getRest(Polynomial p)** which does the same thing, but it returns the remainder of the division.

```
public Polynomial divide (Polynomial p) {
		Collections.sort(this.elements, Collections.reverseOrder());
		Collections.sort(p.elements, Collections.reverseOrder());

		Polynomial result = new Polynomial();
		Polynomial rest = this;
		Polynomial aux = new Polynomial();
		Monom temp = new Monom();
		Monom a = new Monom();
		Monom b = new Monom();

		while(rest.elements.size() != 0 && p.getFirstTerm().getExp() <=
rest.getFirstTerm().getExp()) {
			a = rest.getFirstTerm();
			b = p.getFirstTerm();
			temp = a.divide(b);
			result.addMonom(temp);

			aux.elements.clear();
			aux.addMonom(temp);
			aux = aux.multiply(p);
			rest = rest.substract(aux);
```

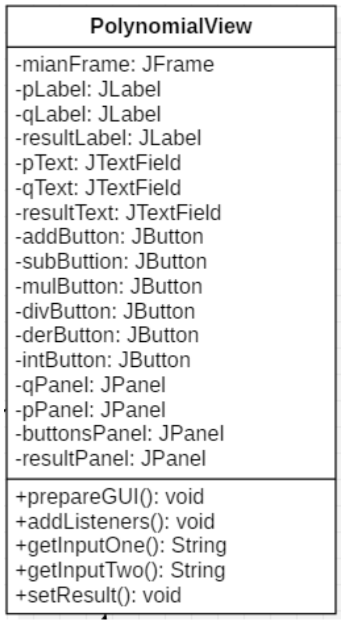
```
            }

            return result;
            }
```

- **public void derivate();** - Differentiation of polynomials
  - This method differentiates **this** polynomial. The class Iterator is used to create an iterator, thus every element is differentiated on the level of monomials, constants are removed.

```
public void derivate() {
            Iterator<Monom> i = this.elements.iterator();
            while(i.hasNext()) {
                  Monom m =(Monom) i.next();
                  if (m.getExp() == 0) {
                        i.remove();
                  }
                  else {
                        m.derivate();
                  }
            }
            }
```

- **Integration** works in the same manner, only there is no iterator used like in **derivate()** method, we iterate with a foreach loop.
- **public void getPolynomial(String input);** - Extracts coefficients and exponents from a string.

4.3 Class PolynomialView



The PolynomialView class implements the view of the application. The Swing library stays at the base of this class and view, it has as attributes swing elements like text fields, labels, buttons or panels.

There is a method called **prepareGUI()** in which every component is prepared, grouped into panels and added to the frame. There are four panels, three of them contain a label and text field (two inputs, one output) and the fourth contains the buttons.

There also is a method in which ActionListeners are added to the buttons and we can find another three methods for getting the strings from the input and giving an output.

4.4 Class PolynomialController

Controller class makes connection between model and view. This class has an attribute PolynomialView view . I chose not to put Polynomial attributes, because in every method new local polynomials are used.

The controller of the class creates and associates ActionListeners for the buttons.

In this class we can find several inner classes, for each button an inner class is created which implements the interface ActionListener and in the method **actionPerformed(ActionEvent e)** the behavior of the button is implemented. In each case this is similar, the controller takes the inputs from the view, calls the corresponding operations from the model, and the view is updated with the result.

There is an example for addition:

```
class AddListener implements ActionListener{
            public void actionPerformed(ActionEvent e) {

                  view.setResult("");
                  Polynomial p = new Polynomial();
                  Polynomial q = new Polynomial();

                  p.getPolynomial(view.getInputOne());
                  q.getPolynomial(view.getInputTwo());

                  Polynomial result = p.add(q);
                  view.setResult(result.toString());

      }
```

where class AddListener is an inner class.

In case of division, if the divider is 0, a message is shown to the user. The implementation involves testing the input, if the input is an empty string, the message is displayed.

```
class DivListener implements ActionListener{

            @Override
            public void actionPerformed(ActionEvent e) {
                  view.setResult("");
                  Polynomial p = new Polynomial();
                  Polynomial q = new Polynomial();

                  if(view.getInputTwo().equals("")) {
                        view.setResult("Please introduce a polynomial, division by
0 is not possible.");
                        return;
                  }

                  p.getPolynomial(view.getInputOne());
                  q.getPolynomial(view.getInputTwo());


                  Polynomial result = p.divide(q);
                  Polynomial remainder = p.getRest(q);
                  view.setResult("Q:" + result.toString() + "         R:" +
remainder.toString());
            }
```

```
        }
```

4.5 Class Main

The Main class is for testing the application, where model, view and controller objects are created. It contains a main() method to start the application.

5. Usage and Testing

User has to introduces the data then by pressing one of the buttons he can choose the preferred operation.

The valid input format is: coefficient + “x^” + exponent.

Here are some valid inputs:

- 1x^2 + 3x^1 + 5x^0
- 3x^2019 + 2x^2018 + 123x^0
- 2x^2 + 4x^0

All things regarding the usage were discussed in previous chapters.

The testing was done by using the JUnit framework using assertions. A class named **TestAssertions** was created and in the method **testAssertions()** some test cases were introduced. Some examples:

```
public void testAssertions() {
        Polynomial p = new Polynomial();
        Polynomial q = new Polynomial();

        p.getPolynomial("1x^3-2x^1+3x^0");
        q.getPolynomial("2x^2-1x^0");
        Polynomial result = q.add(p);

        Polynomial der = new Polynomial();
        der.getPolynomial("1x^3-2x^1+3x^0");
        der.derivate();

        Polynomial integ = new Polynomial();
        integ.getPolynomial("3x^2+1x^1+3x^0");
        integ.integrate();
```

```
		Polynomial x1 = new Polynomial();
		Polynomial x2 = new Polynomial();
		x1.getPolynomial("1x^1+1x^0");
		x2.getPolynomial("1x^1+2x^0");
		
		assertEquals(" + 1.0x^3 + 2.0x^2 - 2.0x^1 + 2.0x^0", result.toString());
		//assertEquals(" + 1.0x^3 - 2.0x^2 - 2.0x^1 + 4.0x^0", 
p.substract(q).toString());
		assertEquals(" + 0.5x^1", p.divide(q).toString());
		assertEquals(" - 1.5x^1 + 3.0x^0", p.getRest(q).toString());
		assertEquals(" + 3.0x^2 - 2.0x^0", der.toString() );
		assertEquals(" + 1.0x^3 + 0.5x^2 + 3.0x^1", integ.toString());
		assertEquals(" + 1.0x^2 + 3.0x^1 + 2.0x^0", x1.multiply(x2).toString());
		
		
		}
```

For running these case another class called **TestRunner** was created, and with the help of JUnitCore all the cases were tested. If all cases are successful, a message is displayed on the console saying “true”, otherwise the errors are displayed.

6. Conclusions

During the development of this assignment, some research was needed. After this assignment I can say that I am more familiar with creating a view manually, before I have done it only with using some drag and drop tools. Also I understood what is the meaning of the MVC pattern and how should I use it and I can see the strength of such a pattern.

During the research I met a lot of new information, it helped me understanding better the structure of some in-built classes.

Another important thing for me was to observe and realize how and where an application can be optimized and what can be developed in the future.

Future developments:

- Optimization of algorithms, unfortunately most of them (except for differentiation and integration) have a time complexity of O(n^2).
- Making the input requirements to be more user-friendly (not forcing to introduce coefficients 1 or exponents 0).
- Handling polynomials with more than one variable.
- Handling exceptions.
- Nicer GUI, dialog windows, greater place for displaying messages and results.

7. Bibliography

- https://www.tutorialspoint.com/junit/junit_using_assertion.htm
- http://www.anidescoala.ro/educatie/matematica/formule-matematice-algebra-liceu-generala/impartirea-polinoamelor/
- https://www.geeksforgeeks.org/how-to-use-iterator-in-java/
- https://docs.oracle.com/javase/8/docs/api/index.html?javax/swing/package-summary.html
- Chifu Viorica: PT_lab1.pdf
- https://stackoverflow.com/questions/28859919/java-regex-separate-degree-coeff-of-polynomial
- https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html
- https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
- https://en.wikipedia.org/wiki/Monomial