Assignment 1

Student: Nagy Hanna
Group: 30421

Propose, design and implement a system for polynomial processing. Consider the polynomials of one variable and integer coefficients.

1. **Objective**

The purpose of the assignment 1 is to design and implement a system which perform some operations on a pair of polynomials or on just one polynomial.
The polynomials are regular ones with only one variable and they can be of any order.
The operations which should have been implemented are the followings: addition of two polynomials, subtraction of two polynomials, multiplication of two polynomials, division with remainder of two polynomials and differentiation and integration (anti derivative) of one polynomial. Additional operations could have been implemented such as the value of the polynomial in a point or if it is possible then to find a root of the polynomials. In my solution of the assignment there is implemented the addition, subtraction, division, multiplication, differentiation and integration (anti derivative).

2. **Analysis of the problem**

The end user or the client which will use this polynomial – processing system will have to able to enter in a specified mode the first polynomial and the second polynomial.
In a way in which the user can enter the first and the second polynomial is to use text fields.
After these actions took place, the end user will have to choose which operation she or he wants. This choosing of the operations can be done on several ways: by radio buttons, by regular buttons or by typing the name of the operation. The simplest way, the cleanest and easiest way for the user to choose the operations is by clicking a regular button. In my version of solution to this problem this choosing is applied. After choosing the operation the user wants. After clicking or choosing in any way the wanted operation the end user/ client should be able to see the result or results. If some error occurs then the error messages.
After one performed operation on a set/ pair or on a single polynomial the user can perform some other operations without entering the same polynomial again and again.
Let's take an example:

Scenario:
The user wants the solution of an addition of two polynomials of any order. Then the user have to write in a special way the polynomial in the text field. We suppose that the client knows how a polynomial looks like and she/he know how to type in and what the notation means. So let's suppose the user want to perform some operations on the polynomials: 2x^2 +3x^1 -3 =0 and 3x^3 +3x^2 +7 =0. Spaces are put between two monomials because it helps us to correctly split the monomials in the program itself. So, the user types these polynomials in the given text fields and he/she wants the result of the addition of these two polynomials. The client clicks on the Addition button and then the result: 3x^3 +5x^2 +3x^1 +4 =0 should be displayed on the third text field which should be un-editable to ensure that the result was given by the calculation of the program and not another user typed in there. Now, we see on the gui (graphical user interface) which is a window containing the operation buttons, the text fields where the

polynomials are written and the text fields where the result of the operations is displayed. The program is written in such a way that the client after getting a result of the wanted operations, can perform another operation on those polynomials without typing them back in the text fields. Now, after the addition of those polynomials, the user also wants to subtract the first polynomial from the second one. Then he/she should click on Subtraction button and the result: -3x^3 -2x^2 -7 =0 should be displayed on the text field. The user have to perform these actions in order to have the result from the other operations like multiplication and division. After clicking on the multiplication button the shown result is:
6x^5 +15x^4 +5x^2 +21x^1 -21=0. In the case of the division the user will see to results. First, the quotient will be displayed on the first result text field and the remainder will be shown on the second result text field which is special created for the remainder.
The other two operations as differentiation and integration (anti derivative) are perfomrmed only on one polynomial, and that is the first one, in means of that which is typed in the first text field. The derivative of the first given polynomials would be: 4x^1 +3 =0 and the anti derivative would be: 2/3 x^3 +3/2 x^2 -3x^1 =0. The coefficients are displayed as doubles.

3. Project

My implementation of the assignment is structured into five packages, namely: gui, mainController, modelElements, operationsPack and testPackage.

3.1 In the gui package, there are implemented the classes needed for the graphical interface for the user like ButtonsPanel, InputOutputPanel and MainFrame.

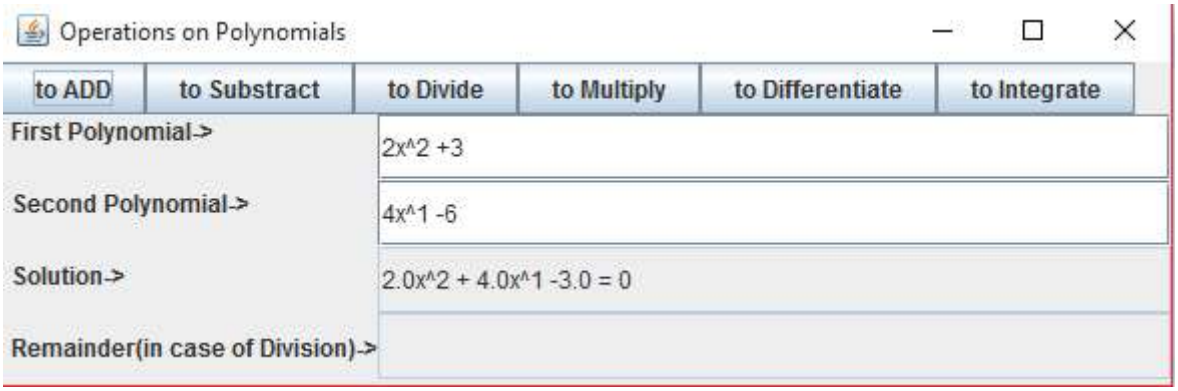


Figure 1

As we can see on the figure 1, the whole window is the so called mainframe, then at the north side of the frame there is the ButtonsPanel, which contains the buttons for choosing the wanted operations and below this panel there are the textfield declared in the InputOutputPanel.

```
/**
 * MainFrame class is JFrame includes the ButtonsPanel and the
 * InputOutputTextPanel
 */

public class MainFrame extends JFrame {

	private static final long serialVersionUID = 1L;
	private ButtonsPanel buttonsPanel;
		private InputOutputTextPanel inputTextPanel;
```

```
/**
 * ButtonsPanel class is a JPanel which contains the
 * buttons for the operations;
 */

public class ButtonsPanel extends JPanel{

	private static final long serialVersionUID = 1L;

	private JButton additionButton;
	private JButton subtractionButton;
	private JButton divisionButton;
	private JButton multiplicationButton;
	private JButton derivativeButton;
	private JButton antiderivativeButton;


		public ButtonsPanel(){
			...
	}



/**
 * InputOutputTextPanel class is a JPanel which contains the
 * textfields where the polynomials
 * will be written and on these will the
 * result/s be displayed
 */

public class InputOutputTextPanel extends JPanel{

	private static final long serialVersionUID = 2244711167508244769L;

	private JTextField firstPolynomInput;
	private JTextField secondPolynomInput;
	private JTextField resultPolynomOutput1;
	private JTextField resultPolynomOutput2;



		public InputOutputTextPanel(){
	}
```

3.2 In the mainController package, there are the controlling classes like the OperationsController which controls the operations, sincronizes the buttons and textfield for choosing operations and displaying the results at their right place on the main frame and there is the mainController class which just contains the essential main method which calls the OperationsController's runOperations method.
In the OperationsController class there are implemented two private inner classes for the use of the actionlisteners.

```
public class OperationsController {

	private MainFrame mainFrame;
	Polynomial p1, p2;
```

```
	public void runOperations( ) {

		mainFrame = new MainFrame( );

		mainFrame. addActionListenerToButtonsPanelButtons(new
OperationsActionListener ( ));
		mainFrame.addActionListenerToInputTextFields(new
InputTextActionListener( ));

	}

		private class OperationsActionListener implements ActionListener
	    {
	}
```

3.3 ModelElements package contains them models/elements needed to solve the proble: Polynomial and CoefficientDegree classes.
In the Polynomial Class we have some methods which helps us to set and get the coefficients, to transform a polynomial into a string in order to display on the textfield, equals() method to compare to polynomials and decide whether they are equal or not, it returns a Boolean value and other methods which are needed to perform the operations.

3.4 OperationsPack contains the classes for the operations. We have an abstract class namely Operations with an abstract method called execute (Polynomial …p1). The other classes in this package like Addition, Subtraction, Division, Multiplication, Differentiation and Integration are subclasses of the abstract class Operations. They each implement the public Polynomial execute (Polynomial …p1) method in their way corresponding to their behaviour.

The implemented division is based on the Eclidean algorithm for the long division with remainder.
Code snippet:

```
package operationsPack;

import modelElements. Polynomial;

public class Division extends Operations {

	Polynomial remainder;
	Polynomial quotient;

	public Division( ) {

	}

	public Polynomial execute ( Polynomial ... p1 ) {
		Polynomial remainder = new Polynomial( p1[0] .getSize( ) );
		// Polynomial quotient = new Polynomial(p1.getSize() - p2.getSize());
```

```
                Polynomial quotient = new Polynomial ( ) ;
                // System.out.println("size of p1: " + p1[0].getSize());
                // System.out.println("size of p2: " + p1[1].getSize());
                for (int i = 0 ; i < remainder.getSize( ); i++) {
                        remainder .setCoeff(p1[0].getCoeff(i), i);
                        remainder. setDegree(p1[0].getDegree(i), i);
                }
                double coef = p1[1].getCoeff(p1[1].getSize() - 1);

                int d = p1[1].getSize() - 1;
                Addition newAdd = new Addition();
                Subtraction newSub = new Subtraction();
                Multiplication newMult = new Multiplication();
                Polynomial resMult = new Polynomial();

                while (remainder.getSize() - 1 >= d) {

                        double newCoeff = (remainder.getCoeff(remainder.getSize() - 1)) /
coef;
                        int newDegree = remainder.getSize() - 1 - d;

                        Polynomial aux = new Polynomial(remainder.getSize() - d);
                        aux.setCoeff(newCoeff, newDegree);
                        aux.setDegree(newDegree, newDegree);

                        quotient = newAdd.execute(quotient, aux);

                        resMult = newMult.execute(aux, p1[1]);

                        remainder = newSub.execute(remainder, resMult);

                }
                for (int i = 0; i < remainder.getSize(); i++) {
                        p1[2].setCoeff(remainder.getCoeff(i), i);
                        p1[2].setDegree(remainder.getDegree(i), i);

                }
                return quotient;
        }

}
```

Code snippet 1

As above were said, the classes corresponding to the actual operations (addition, subtraction, division, …) extend the abstract class Operations which has its implementation seen at the Code snippet 2.

```
package operationsPack;

import modelElements.Polynomial;
```

```
public abstract class Operations {

        public abstract Polynomial execute(Polynomial ...p1);


}
```

Code snippet 2

3.5 The testOperation pack with only one class namely TestOfOperations has the responsibility to test the operations implemented in the classes with the operations.

```
package testPackage;


/**
 * Testing the operations
 */
public class TestOfOperations {

   Polynomial polynom1;
   Polynomial polynom2;
   Polynomial expectedResult;
   Polynomial actualResult;
   Polynomial expectedRemainder;
   Polynomial actualRemainder;

   Operations operation;

   /**
    * polynom1 = 2 x ^ 3 + 3 x ^ 2 -4 polynom2 = 1 x ^ 1 -3
    */
   public TestOfOperations ( ) {
            polynom1 = new Polynomial ( ) ;
            polynom2 = new Polynomial ( ) ;
            expectedResult = new Polynomial ( ) ;

            polynom1.addCoeffDegree ( new CoefficientDegree ( -4,  0 ) );
            polynom1.addCoeffDegree ( new CoefficientDegree ( 0, 1 ) );
            polynom1.addCoeffDegree ( new CoefficientDegree ( 3, 2 ) );
            polynom1.addCoeffDegree ( new CoefficientDegree ( 2,  3 ) );

            polynom2.addCoeffDegree ( new CoefficientDegree ( -3, 0 ) );
            polynom2.addCoeffDegree ( new CoefficientDegree ( 1, 1 ) );

   }

   /*  *
    * result = 2 x ^ 3 + 3 x ^ 2  + 1 x ^ 1 - 7
    */
```

```
    @Test
    public void testAddition ( ) {

            operation = new Addition ( ) ;

            expectedResult. addCoeffDegree(new CoefficientDegree ( -7 ,  0 ) ) ;
            expectedResult. addCoeffDegree(new CoefficientDegree ( 1, 1 ) ) ;
            expectedResult. addCoeffDegree(new CoefficientDegree ( 3 ,  2 ) ) ;
            expectedResult. addCoeffDegree(new CoefficientDegree ( 2 ,  3 ) ) ;

            actualResult = operation. execute ( polynom1, polynom2 ) ;
            assertTrue  ( " addition result " , expectedResult.equals ( actualResult ) )
;
    }

    /* *
     * result = 2 x ^ 3 + 3 x ^ 2  -1 x ^ 1 - 1
     */

    @Test
    public void testSubtraction ( ) {

            operation = new Subtraction ( );

            expectedResult. addCoeff Degree ( new CoefficientDegree ( -1 ,  0) ) ;
            expectedResult. addCoeff Degree ( new CoefficientDegree (- 1, 1 )  ) ;
            expectedResult. addCoeff Degree  ( new CoefficientDegree ( 3,  2 ) );
            expectedResult. addCoeff Degree ( new CoefficientDegree ( 2,  3 ) );

            actualResult = operation. execute( polynom1, polynom2 ) ;

            assertTrue ( " subtraction result " , expectedResult. Equals (
actualResult ) ) ;
    }
```

Code snippet 3 .

I have a separate method for all the operations being tested. I used two known polynomials and their result after every operation. Then, in the corresponding method I called the corresponding operation class and then I used the assertTrue(String mess, Boolean state) method to decide wheather the actual result and the expected (known) result are the same (they are equal). The equality of two polynomials is verified by the equals() method (below, code snippet 4) implemented in the Polynomial class.

```
public boolean equals ( Polynomial p1 ) {
                boolean equal = true;
                int i;
                i = 0;
                al: while ( equal && ( i <  this. getSize ( ) ) ) {

                        if (this. getSize (  ) ! = p1.getSize ( ) ) {
                                equal = false;
                                break;
                        } else {
                                for ( Coefficient Degree coef_ deg : p1. pairs) {

                                        if  ( coef_deg.getCoefficient ( ) ! = this . getCoeff ( i ) ) {
                                                // System. out. println( " they are not equal " +
coef_ deg. Get Coefficient( ) + " " + this. Get Coeff( i ) );
                                                equal = false;
                                                break al;
                                        }

                                        i + +;
                                }
                        }
                }
```

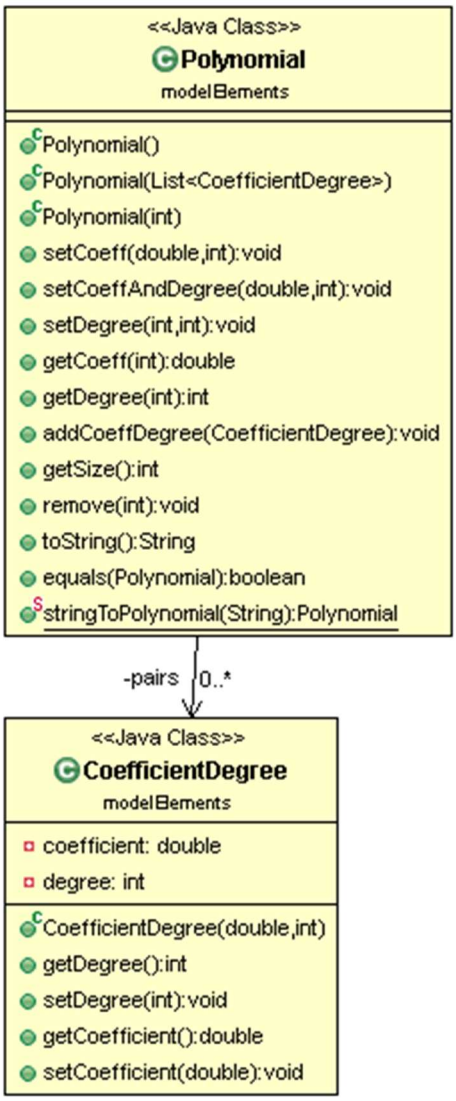
Code snippet 4

Relations. UML diagrams.

Diagram snippet 1:

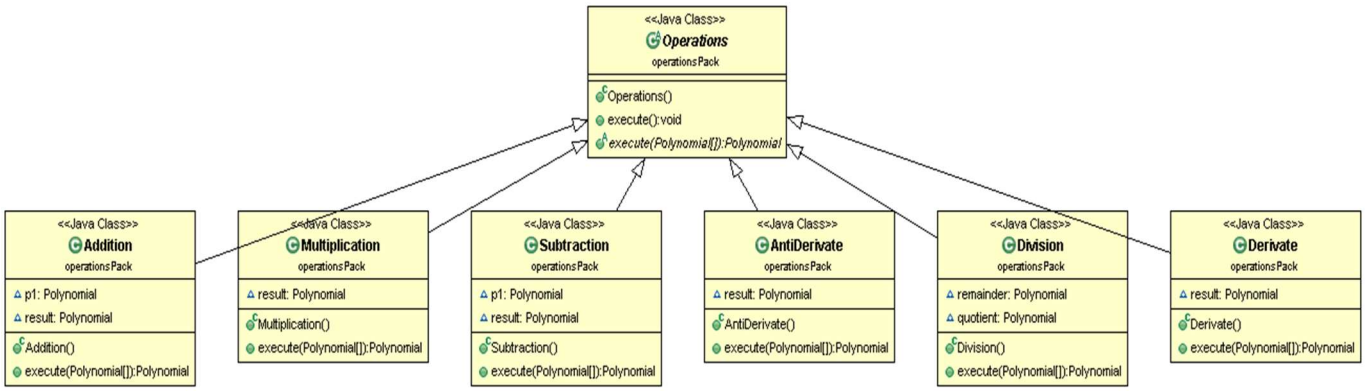
This is a part of the UML diagram of the whole system. This one represents the relations between the classes in the gui package needed to define the graphical interface of the user where he/she could type the polynomials and also see the results.
As above at the description of the packages were said that the mainFrame class has the role, as its name said, of build and display the gui and it contains 2 main panels one for the operation Buttons and one for to input/output polynomials and its result.

Diagram snippet 2:



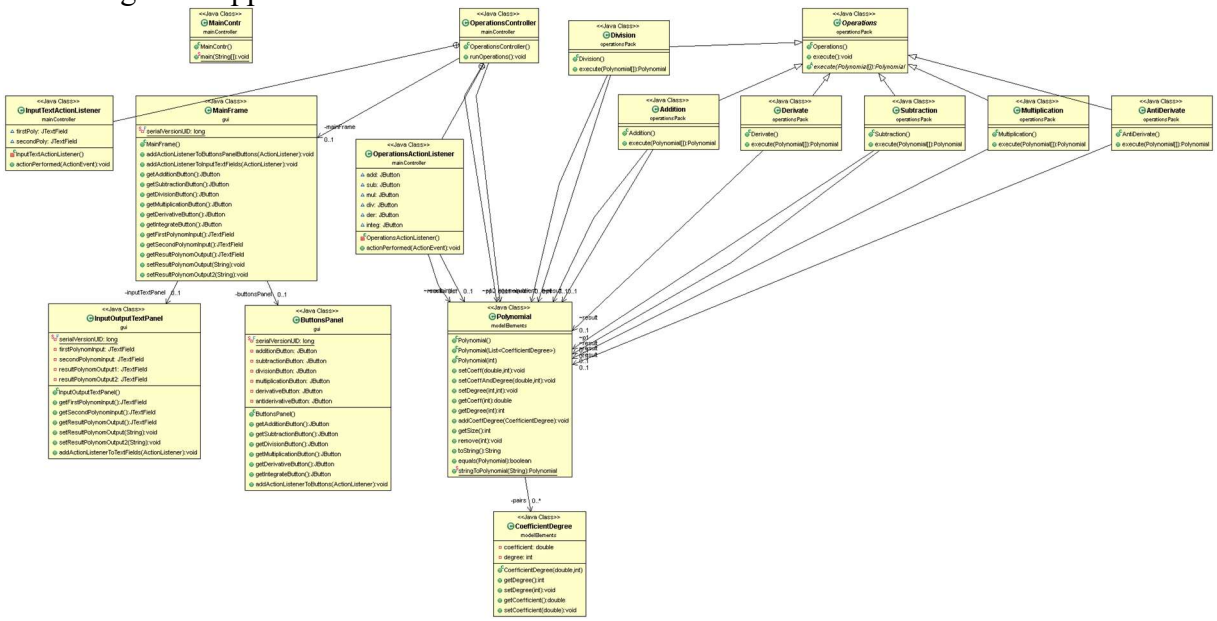
These classes are the two parts of the modelElement package. As in the real world, a polynomial in build up by monomials. In this relation, the monomial is represented by the CoefficientDegree class which has two instance variables, one for the coefficient and the other one is for the degree. The coefficient is the type double, because at the generating the integral (anti derivative) of the polynomial there is the case of division of the coefficient the power of the monom.

Class Diagram snippet 3:



As we can see above we have an abstract class Operations, which is a generalization of the operations. This Operation class can be meant of any operation which can be performed with the regular polynomials. It is defined abstract because we do not want to be instantiated (example Operations exampleOp = new Operation ( ) – as it was said in this way we cannot instatiate the Operation class), but we can use the polymorphism induced by these relations, and we can declare (just declare) an object with the type Object and then we instantiate to a class which extends the abstract class like Addition, Subtraction, Division, Multiplication, Differentiation and Integration. For example, we use this concept in the testOperationsPack in the testOperations class in this way: Operation result; and in every method which tests the operation we initialize of the type of the operation we test. For example: Operations result = new Addition().
Every sub class of the abstract class Operations implement the the method public Polynomial execute(Polynomial... p1) with the one, two or three parameters depending on the operation being performed. Addition is performed with two polynomials p1[0] and p1[1] and the result Polynomial is returned back. Subtraction, Multiplication behave in the same way as the Addition. In case of the Division we pass three parameters p1[0] and p1[1] for the polynomials and the p1[2] is for the remainder. Differentiation and Integration uses just one polynomial. As we observed, this three-dot strategy is very useful in this case, when we have to implement the very same method but with different parameter number.

Class diagram snippet 4:



This is the whole class diagram.
He operation classes: Addition, Subtraction, Division, Multiplication, Differentiation and Integration are in has-a relationship with the class named Polynomial, so-called association. Because they define and instantiate one or two polynomials. Between the Polynomial and Coefficient class there is a relation called composition, because if we distroy the Polynomial then there will not be present any coefficient or degree without that polynomial.
In the mainFrame Class we define two classes, namely: ButtonsPanel and InputOutputPanel. We can say that these panels could exists without the the mainFrame class, so between these elements: the mainFrame and the two panels: ButtonsPanel and InputOutputPanel there is an aggregation relation.

4. Implementation and testing

For the implementation, the object oriented way was developed. It is used the advantages, main principels of the object oriented programming. For example, it is used the encapsulation principle in the implementation. Some details are made private and hidden from the other classes. Then, it is used the abstraction of the object oriented programming. There is an inheritance tree between the Operations class, which is a generalized class and its subclasses are the actual operations: Addition, Subtraction, Division, Multiplication, Differentiation and Integration. In the testing class it is used the polymorphism induced by these relationships of these classes. Classes and objects as real world entites are defined in the implemented system.

Testing was made using the testing framework so called Junit. There is just one class named TestsOfOperations which has methods named test<operation being tested> for every operation: testAddition, testSubtraction, testDivision, testMultiplication, testDivision, testDerivate, testIntegration.

```
@Test
        public void testAddition(){
        ...
    }
@Test
        public void testSubtraction() {
        ...
    }
@Test
        public void testMultiplication() {
        ...
    }

@Test
        public void testDivision() {
        ...
    }
@Test
        public void testDerivate() {
        ...
    }
@Test
        public void testIntegration() {
        ...
    }
```

In the constructor of the TestsOfOperations, I declared two polynomials to be known for the testing, and in every testing method there is an actual result and the expected result. Using assertTrue(String message, Boolean bool) the two results are compared by the polynom1.equals(polynom2) method defined in the Polynomial class.

5. Conclusion. Further implementation.

This assignment combines the object oriented programming principles and also some real world issues or frequently needed informations. The polynomials are used all over the computational systems and so on.
The implementation is made in a way that if someone wants to modify, to add, to remove, to extends and to maintain then these operations to be easier to done. If the programmer wants to add a new operation which is not in the implemented system

then he or she need to create a very new class to extend the abstract class Operations and implement the public Polynomial execute (Polynomial ...p1) method. If this given method does not correspont to the specification of the new operation the programmer wants then that newly created class does not have to extend the abstract class Operations, but this just in a very peculiar case.

6. Bibliography

Division of the polynomials (Euclidean algorithm)
https://en.wikipedia.org/wiki/Polynomial_greatest_common_divisor

Use of Junit https://www.youtube.com/watch?v=AN4NCnc4eZg