*Technical University of Cluj-Napoca*

*Faculty of Automation and Computers*

*Department of Computer*

*2015 - 2016*

**Programming Techniques**

**Homework 1**

**Student: Jucan Diana**

**Group: 30421**

Content

1. Objective. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **3.**
2. Problem analysis
   1. Problem Specification. . . . . . . . . . . . . . . . . . . . . . . . . . **3.**
   2. Problem Modeling. . . . . . . . . . . . . . . . . . . . . . . . . . . . . **4.**
   3. Scenarios and Use Case . . . . . . . . . . . . . . . . . . . . . . . . .**5.**
3. Design
   1. Relational Diagram. . . . . . . . . . . . . . . . . . . . . . . . . . . . . **6.**
   2. Packages and Classes . . . . . . . . . . . . . . . . . . . . . . . . . . **7.**
   3. User Interface. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **11.**
   4. Data Structures . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .**13.**
   5. Relations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .**13.**
   6. Algorithms. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .**13.**
4. Testing the application. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **15.**
5. Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **16.**
6. Bibliography. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **16.**
7. **Objective**

The objective of the current project is to implement a system for polynomial processing. The student must transpose the real world objects ( in our case these are not proper objects) into classes. In this way, we get the Object Oriented Programming principle called abstraction.

We also need to pay attention to all the principles of Object Oriented Programming: abstraction, encapsulation, inheritance and polymorphism, this being the objective of the project.

1. **Problem Analysis**

**2.1 Problem Specification**

(EN) Lab

Homework 1

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

The program will be able to perform some operations that can be done on one or two polynomials. Among these operations there are:

* Addition of two polynomials
* Substraction of two polynomials
* Multiplication of two polynomials
* Division of two polynomials
* Integration of a polynomial
* Differentiation of a polynomial

The program will have an user interface, where the user will introduce the polynomials in their mathematical form, for example 5x^2 + 3x^1 – 2, and according to some buttons corresponding to the desired operation ( those mentioned above ) , on the frame will be displayed the result.

* 1. **Problem Modeling**

Problem modeling means dividing a given problem in relevant classes. In our case, we have designed it in such a way that we have four packages (polynomial.gui, polynomial.main, polynomial.operations, polynomial.polynomial) and many classes. The main classes are : Operations , MyFrame, Polynomial, Term and Main.

Because we are talking about an obiect – oriented programming language, we divided the class Operations into other classes: Addition, Substraction, Multiplication, Integration, Division, Differentiation and Division. We did this making the class Operations to be an abstract class and having one method named execute. Here might have appeared some conflicts, because the method execute has as parameters one or more polynomials, depending on the operation. For example, if we want to add two polynomials, we need two parameters, but in case of integration, we need one single polynomial. This can be solved writing the header of the method in the following way:

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

We made the method public for calling it outside the Operations class. It is an abstract method because there is no implementation of the method in the Operations class, but every sub-class must implement the method. Polynomial means that *execute* returns an object of type Polynomial. As formal parameters we have declared a Polynomial p1 preceded by three dots, this means that the method expects to receive as parameters one or more polynomials.

We know from mathematics that a polynomial is made of monomials.

For example, the polynomial is 5x^3 + 2x^2 -7 and a monomial is 2x^2. A monomial has a coefficient, 2, and a degree. Every monomial has this form, a pair of coefficient – degree, so we can design a class called Term to retain this pair. Then, the polynomial is a list of monomials, so we create a class Polynomial to handle this relation.

In MyFrame class we have created the user interface, which contains 6 buttons, one button for every operation, a button to compute the polynomials, meaning to transpose them into that list of monimials and some fields where we introduce the polynomials and where the result is displayed.

* 1. **Scenarios and Use Case**

The project that processes the polynomial can be used to make fundamental operations on polynomials. A scenario where the user wants to add the polynomials and then to substract them is handled, because their addition is not performed and then returned as a new value of the current polynomial, but the result is put into another variable. In this way, we can add the polynomials, and them substract them or whatever we want, because the are no changes made by the application on the structure of the monomials.

An example of working will be given:

We assume that the user has entered the polynomials:

12x^4 - 6x^3 + 2x^1 + 7 and 3x^2 + 2

The button Compute polynomials must be pressed in order to associate to some 2 Polynomial objects, the corresponding monomials ( and to every monomial to set its degree and its coefficient ). Then we have six options, and the desired button will be pressed:

* Addition result: 12x^4 - 6x^3 + 3x^2 + 2x + 9 ;
* Substraction result: 12x^4 - 6x^3 - 3x^2 + 2x + 5 ;
* Multiplication result: 36x^6 - 18x^5 + 24x^4 - 6x^3 + 4x + 21x^2 + 14 ;
* Division result:

Quotient: 4x^2 - 2x -2 ;

Remainder: -2x^2 +6x +11 ;

* Differentiation: 48x^3 - 18x^2 + 2 ;
* Integration: 12/5x^5 - 6/4x^4 + x^2 + 7x + c ;

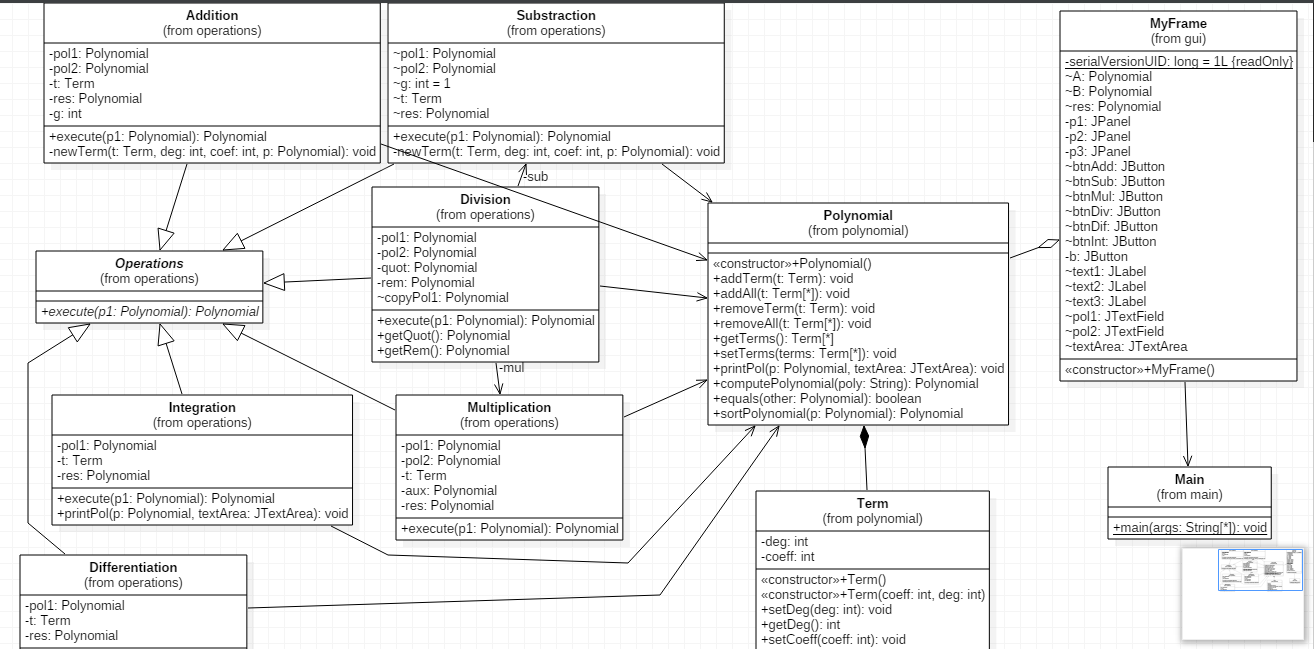
We notice that on integration, we have a constant c.

A scenario could be if we want to raise a polynomial to a certain power. In this case, we make successive multiplication of the polynomial with itself. We also can make the multiplication of a polynomial with a scalar, by introducing the first polynomial in the corresponding JTextField, and the scalar in the second JTextField. The scalar will be interpreted as cx^0, where c is the scalar.

1. **Design**

**3.1 Relational diagram**

This is the relational diagram of the project:



We have a relation of composition between Polynomial and Term, and a relation of aggregation between Polynomial and MyFrame.

Another relation is Association, where objects of the two classes can know about each other, but they do not affect each others lifetime. The objects can exist independently and which class A object knows about which class B objects can vary over time. Any relationship between instances of two classes, where an instance of one class needs to know about an instance of the second class in order to perform it's work - is an Association relationship.

We have associations between:

- Division and Multiplication, Division and Substraction;

- every subclass of Operations and Polynomial;

- every subclass of Operations and Term;

- every subclass of Operations and MyFrame;

- MyFrame and Main;

We have also an extends relation between Operations and: Addition, Substraction, Multiplication, Integration, Differentiation, and Division.

**3.2 Packages and Classes**

First of all, we want to make clear what a Java class and a package are. For that, we will give an example that can be understood by everyone.

In the real world, you'll often find many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that your bicycle is an *instance* of the *class of objects* known as bicycles. A *class* is the blueprint from which individual objects are created.

A package is a namespace that organizes a set of related classes and interfaces. Conceptually you can think of packages as being similar to different folders on your computer. You might keep HTML pages in one folder, images in another, and scripts or applications in yet another. Because software written in the Java programming language can be composed of hundreds or thousands of individual classes, it makes sense to keep things organized by placing related classes and interfaces into packages.

This being said, there will be presented the packages and the classes of the project.

There are four packages: *polynomial.gui, polynomial.main., polynomial.operations* and *polynomial.polynomial*. As you may observe, the name of the packages follow the java naming conventions, according to that the names are lowercase letters, the first character being a letter ,not a digit or a special character. The first part of the name is actually the name of the project, followed by a dot and then by the proper name of the package. For example, *polynomial.gui* is a package that contains a single class, MyFrame, responsible with the the graphics, the user interface. The class MyFrame will be described later.

From the name of the package we can deduce the functionality of its classes. Polynomial.main package suggest the existence of a class that contains the main function. Actually, that class is named Main. It will be presented the syntax of the class Main:

**package** polynomial.main;

**import** polynomial.gui.MyFrame;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

**new** MyFrame();

}

}

The first line in a class definition is the package where the class belongs. In our case, the package is polynomial.main. The instruction is followed by semicolon as every instruction in programming language. Next there is the imported packages section. If in our class we want to have an instance of a class from another package, we need to import that package. In the main function we created a new instance of the class MyFrame, so we imported the package containing the class ( import polynomial.gui.MyFrame; ) .

It follows the keyword class preceded by an access modifier, public, and followed by the name of the class. A class may be declared with the modifier public, in which case that class is visible to all classes everywhere. If a class has no modifier (the default, also known as *package-private*), it is visible only within its own package. Inside the brackets, we can put method definitons variables and constants definitions. We have the main method between brackets, which means that the application will be launched from this point. In the main function, we have only one instruction, a new instance of the class MyFrame: new MyFrame ( ); *new* is a keyword. Now the instructions from the MyFrame class will be executed.

The third package is polynomial.polynomial. This contains two classes that are related each other: Polynomial and Term.

**Term class**

Term class is a public class of the package polynomial.polynomial. We know from mathematics that a polynomial is a collection of monomial, and that every monomial has a degree and a coefficient. We choose to name the class Term, it seems more logic . As we said, the class Term contains two instance variables, deg and coeff.

Instance variables are declared in a class, but outside a method, constructor or any block. Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.

The variables deg and coeff are private variables, because we don’t want that another class to change their values, but we can control their values by having getters and setters. We will give example for deg variable.

**public** **void** setDeg(**int** deg){

**this**.deg=deg;

}

This is a setter method. It receives as a parameter a variable of type int, and assignes this value to the current value of the deg variable. The current value of a variable is represented by coding *this.name\_of\_the\_variable*. To better understand the class Term, an example will be given:

Suppose we have the polynomial 5x^3 + 7x^2 – 1x^1 + 4. A term will be 5x^3, and more over, the degree of the term will be 3, the coefficient of the term will be 5. The method getDeg() that is a getter method, helps other classes to get the current value of the deg variable. This can’t be done just by calling the variable instead of calling the method, because the variable has a private modifier. The syntax of the getter is:

**public** **int** getDeg(){

**return** **this**.deg;

}

We have also getter and setter for coeff variable.

A **constructor** is a bit of code that allows you to create objects from a class. You call the constructor by using the keyword **new**, followed by the name of the class, followed by any necessary parameters. Term class contains two constructors. The first one has no parametres, and the second one has two parameters, int coeff and int deg. This values will be assigned to the current values of the instance variables in the following way:

**public** Term(**int** coeff,**int** deg){

**this**.deg=deg;

**this**.coeff=coeff;

}

As we can observe, a constructor has the same name as the class it belongs. It’s like a method but is has no return type;

**Polynomial class**

Polynomial class contains a list of Term objects. In the imported classes section we have:

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.StringTokenizer;

**import** javax.swing.JTextArea;

ArrayList and List are used to mark that a Polynomial class is a collection of Term objects. We have this by coding:

Private List<Term> terms;

and in the constructor :

terms = new ArrayList<Term> ( );

This means that whenever we make an instance of the Polynomial class, a list of Term, terms, will be automatically created.

As method, there are addTerm and removeTerm, that receive as parameter an object t of type Term, and adds it or removes it from the current list of terms, called terms.

We can also have the possibility of removing an entire list from the current list. This means that every term from the list received as parameter will be removed from the current list (terms). Similarly in addAll method, every term from the list received as parameter will be added to the current list. Suppose that the list as parameter is *List<Term> t* , and the list of the class is called terms, we add or remove all from list t by coding:

terms.addAll (t);

terms.removeAll (t);

Because the terms variable is private, we have getter and setter for it.

**import** java.util.StringTokenizer is useful in the method *computePolynomial*.

Its header is:

public Polynomial computePolynomial (String poly)

The method is public because it is called from the outside of the Polynomial class. It returns an object of type Polynomial and receives as parameter a string. When the user introduces the polynomial, he actually introduces a string of characters, and this string have to be manipulated. We do this with the help of *computePolynomial* method. Inside the method, it is declared a vector of maximum 1000 integer values, where we will retain all the numbers from the string.

We have an instance of the StringTokenizer class, named *tokens*, which will divide our string *poly* into substrings. This division will be done by the characers “ x^” , this meaning that whenever we meet a space or a “x^”group, what follows after will be put in the string *theToken*. While *tokens.hasMoreTokens ( )* method returns true, *theToken* stringwill receive another token that will be converted to an int value by the method *Integer.parseInt ( theToken )* ;

Now the vector numbers contains all the numbers from the user string. It follow to set this numbers to be the degree and the coefficient of terms composing a polynomial. We make this by creating a new Term and coding:

Polynomial A = **new** Polynomial();

Term t;

**for** (**int** j = 1; j < i; j = j + 2) {

t = **new** Term(numbers[j], numbers[j + 1]);

A.addTerm(t);

}

**return** A;

At this point, the method will return a Polynomial obtained from a string.

Another method inside the Polynomial class is *printPol*. It receives as parametres a polynomial, of course, and a JTextArea, where the polynomial will be displayed. We put a lot of conditions in the printing method, to get a nice representation of the string. For example, if we have a 0 coefficients , that term will not be displayed, if we have a 0 degree that term will contain only the cefficient, or if we have x to the power 1, we will display just “x” instead of “x^1”.

**Operations class**

Operations class, toghether with its subclasses : Addition, Differentiation, Integration, Substraction, Multiplication and Division, compose the *polynomial.operations* package. Operations is an abstract class and has one method named execute. Here might have appeared some conflicts, because the method *execute* has as parameters one or more polynomials, depending on the operation. For example, if we want to add two polynomials, we need two parameters, but in case of integration, we need one single polynomial. This can be solved writing the header of the method in the following way:

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

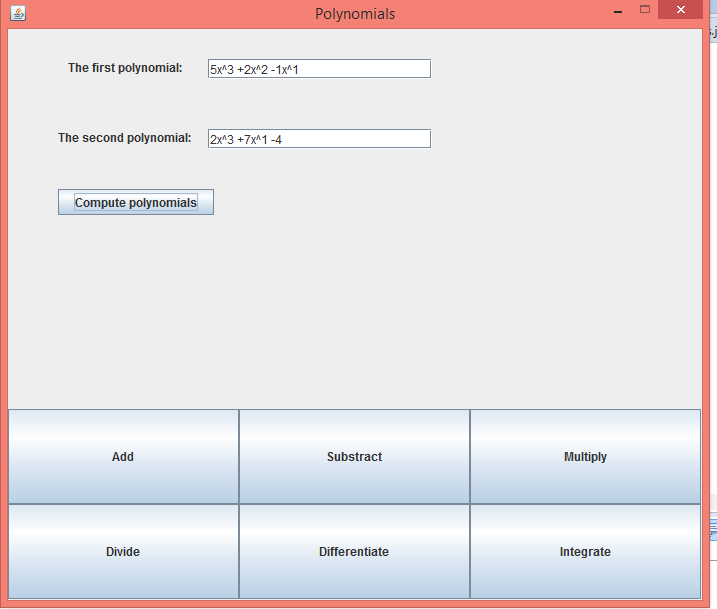
We made the method public for calling it outside the Operations class. It is an abstract method because there is no implementation of the method in the Operations class, but every sub-class must implement the method. Polynomial means that *execute* returns an object of type Polynomial. As formal parameters we have declared a Polynomial p1 preceded by three dots, this means that the method expects to receive as parameters one or more polynomials.

The classes Addition, Differentiation, Integration, Substraction, Multiplication and Division all implements the method execute from their superclass Operations. All the sublasses contains only the execute method and eventually a method called newTerm, for avoiding the repetition of the code when we create many terms for the same polynomial, but Integration class contains also a print method. This is useful when we print an integrate polynomial. For example, if we have the polynomial 5x^3 -2x^4, it is nice not to store the coefficient of the first term after integration as the value of 5/3, but to print effectively 5/3. Some improvements of the method are that we will not print something like 4/1, or 4/4, but we will print 4 and respectively, 1.

**3.3 User Interface**

The user interface is quite simple. It is a JFrame that contains 3 JPannel, organized as a GridLayout of 3 lines and one column. Now, every JPannel has a different scope. The first JPannel has a layout set as null, because we want to organize the components after the coordinates, x and y.

So, the first JPannel contains two JLabels, with the text “The first polynomial: ” and “The second polynomial: ” as in the image:



As we can see, we have 2 JTextFields where we introduce the polynomials, and then we will click Compute polynomials button. This button has an ActionListener according to that when clicked, the method computePol from the Polynomial class will be called and a Polynomial object will be returned.

The second JPannel will contain the results of the operations, and the third JPannel contains 6 buttons, organized as a GridLayout of 2 lines and 3 columns. Every button has an ActionListener; when clicked, the corresponding execute method will be called and the result will be displayed in the second JPanel.

**3.4 Data Structures**

The main data structures used in the project are ArrayList and vectors.

ArrayList was used in the Polynomial class, to mark that a polynomial is a collections, so a list of terms.

Vectors were used whenever we want to have a group of related variables, meaning a vector of integer numbers, of double numbers of a vector of strings.

**3.5 Relations**

The relations of the project were explained in details in the relational diagram section.

The main relations in the project is between the abstract class Operations, and its sublasses Addition, Substraction, Multiplication, Integration, Differentiation and Division. This is an extends relation.

There are some dependencies between classes, that can be visualized in the relational diagram (UML), at 3.1 section, page 6.

**3.6 Algorithms**

In the project we can say that we have several algorithms.

* The algorithm for addition:

The method execute receives two polynomials as parameters. With two for instructions, we go through every term in the first polynomial, and we try to find its pair, meaning a term with the same degree. If this is true, a new term having as the coefficient the sum of coefficients of the first and second polynomial, and as degree, the degree of the current term. In this way, we add to the result the terms with same degree from pol1 and pol2 polynomials.

Now, it remains to add to the result the terms from pol1 and pol2 that has no pair, no term with the same degree in the other polynomial.

* The algorithm for substraction:

The algorithm for substraction is similar with that for addition, except that the new term added to the result could have the coefficient:

* the difference of the coefficients for that term from the first polynomial that has a pair in the second polynomial ;
* the same, if the term is from the pol1 and has no pair in pol2 ;
* 0-coefficient, if the term is from pol2 and has no pair in pol1 ;
* The algorithm for multiplication:

With two for instructions, we go through every term in the first polynomial and multiply it with every term in the second polynomial. The new term will have the degree the sum of the degrees of pol1 and pol2, and as coefficient, the multiplication of coefficients of pol1 and pol2.

After that, it remains to add the terms from the result that has the same degree.

* The algorithm for differentiation:

The algorithm is that for every term of the polynomial, we set the coefficient as being the product between the current coefficient and the current degree, and set the degree as being the current degree-1.

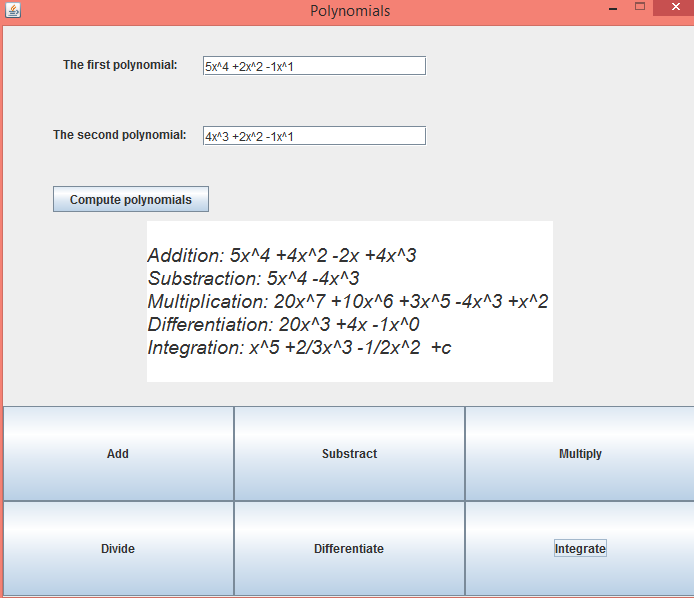
* The algorithm for integration:

The algorithm is that for every term of the polynomial, we set the coefficient as being the ratio between the current coefficient and the current degree+1, and set the degree as being the current degree+1.

* The algorithm for division:

The polynomial pol1 and pol2 are ordered descending after degrees. The division of pol1 to pol2 is done. The first term of pol1 is divided by the first term of the second polynomial, pol2. The result is multiplied by the pol2, and this new result , substracted from pol1, is the first remainder. This steps will be done, by making the first remainder to be the pol1 now. The algorithm is finished when the remainder is less than the pol2.

1. **Testing the Application**



In the previous image, we tested the application.

In order to use the application, we launch the application from the Main function. This will open a window where the user can enter the desired values and selecting the operations by pressing one of the 6 buttons.

Rules for input data:

* The two polynomials must be given in their mathematchical form, for example

5x^3 -5x^1 +2;

Note: between two terms it need to be a white space, because of parsing condition.

* The coefficient must be written even if it is 1.

1. **Conclusions**

The project of polynomials processing helps to think like an object-oriented programmer. It is easy to implement the polynomials as vectors, where the index of the vector represents the degree of the terms, but it is not OOP. So the project come to help to get used with some principles.

For a better performance, there should be implemented cases where exceptions can occur and the application stops working due to an error made by the user. Another thing could be to improve the display, to make it more elegant.

1. **Bibliography**

<http://stackoverflow.com/>

<http://www.purplemath.com/modules/polydiv2.htm>

<http://users.utcluj.ro/~jim/OOPE/>