*Technical University of Cluj-Napoca*

*Faculty of Automation and Computers*

*Department of Computer Science*

*Semester 2015-2016*

***Programming Techniques***

***Homework 1***

***Student: Sporis Ioan Catalin***

***Group: 30421***

***Contents***

1. **Problem specification**
2. **Example of working**
3. **Design**
   1. **Relational Diagram**
   2. **Classes Design**
      1. **CoeffDegree**
      2. **Polynomial**
      3. **Operation**
      4. **UnaryOperation**
      5. **Differentiation**
      6. **Integration**
      7. **BinaryOperation**
      8. **Addition**
      9. **Subtraction**
      10. **Multiplication**
      11. **Division**
      12. **UpperPanel**
      13. **LowerPanel**
      14. **MainWindow**
      15. **BtnsHandler**
      16. **StartApplication**
   3. **Packages**
      1. **polynomial\_op.entities**
      2. **polynomial\_op.operations**
      3. **polynomial\_op.gui**
      4. **polynomial\_op.entities**
      5. **polynomial\_op.test**
4. **Results**
5. **Conclusions**
6. **References**

**1.Problem Specification**

**Homework 1**

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

The program is able to perform some basic operations on one, or two polynomials. Among these, you would find:

* Arithmetic operations: Addition, Subtraction, Division, Multiplication (Binary Operation – on two polynomial)
* Integration, Differentiation (Unary Operations – on one polynomial)

The application has a friendly interface, more like a pocket calculator, where the user is able to type the desired polynomials and then select the operation they want to perform on those. User has to introduce each polynomial in a certain format, typing the coefficient, followed by an small “x”, the character “^”, the corresponding degree followed be a space between the next monomer. The result will appear on the upper display.

.

**2.Analyzing the problem. Modeling the application. Scenarios. Use cases**

**2.1. Analyzing the problem and modeling the application**

Starting from the basic math knowledge, the application is designed to perform some of the basic operations over two polynomials and also, over one polynomial. Some examples would be:

* ***Polynomial addition***

Degree (A + B) <= max(degree A, degree B)

* ***Polynomial subtraction***

Degree (A-B) <= max(degree A, degree B)

* ***Polynomial multiplication***

Degree( A\*B ) = degree( A ) \* degree( B )

* ***Polynomial division***

Degree (A/B) = degree ( A ) – degree ( B )

* ***First order differentiation***
* ***First order integration***

An essential parameter by defining the polynomial is the set of its coefficients. In our case the coefficients follow the integer axis, with the exception of the coefficients that result by performing the division and integration where the coefficients will be of type float. However, this application is set to cast to integer any float value that might result from one of these two operations.

**2.2. Scenario and use cases**

Having the interaction with the system defined some concrete examples of introducing the polynomials are the polynomial: and the polynomial: will have the inputs 12x^4 -6x^3 +2x^2 +7x^0 respectively 3x^2 +2x^0.

The user has to take into account some requirements in order not to trigger some errors. Below are presented a few of them:

* Forgetting to introduce the required data. For instance, binary operations should have two polynomials in order to be able to compute the result
* Another error might be using an incorrect syntax. For example, each pair of CoeffDegree should have a space between one another

**2. Example of working**

The application works on integer coefficients, so, even for division and integration, where there exists the possibility of obtaining values of the type float, it will cast each result to the closes integer. For an unary operation, user has to enter only one polynomial, since the calculations are done on the first input only. If someone needs a different polynomial, they should first press the “Clear” button, in order to delete the last value.

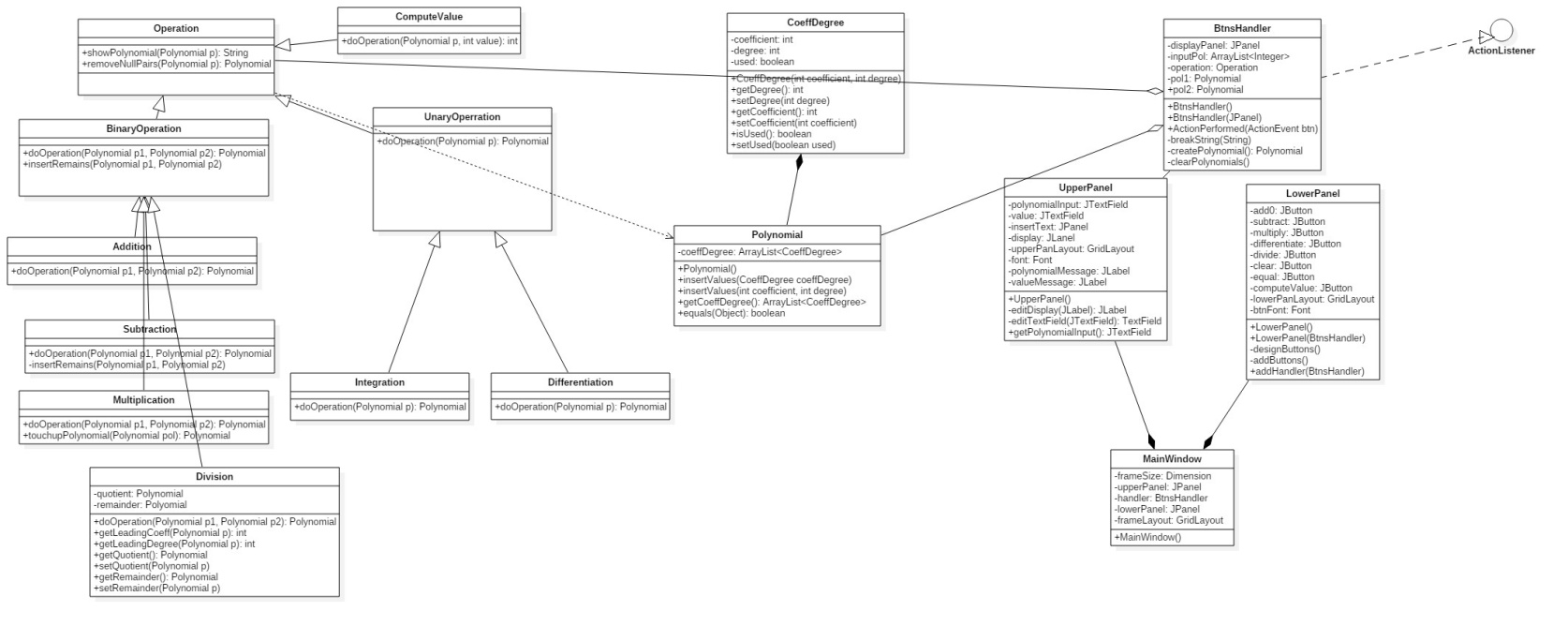
For instance, if the user would introduce as the first polynomial 12x^4-6x^3+2x^2+7x^0 and as the second polynomial 3x^2+2, the results would be the following:

* Addition: 12x64-6x^3+5x^2+9x^0
* Subtraction: 12x^4-6x^3-1x^2-5x^0
* Multiplication: 36x^6-18x^5+30x^4-12x^3+25x^2 + 0x^1 +14X^0
* Division: - Quotient: 4x^2-2x^1-2x^0

-Remainder:-2x^2+6x^1+11x^0

* Integration: 2x^5-1x^4+0x^3
* Differentiation: 48x^3-18x^2+4x^1

The application’s interface is very easy to be comprehended having a display, showing the result, a text field where the desired input is provided, and a set of buttons for the operations that are performed on the polynomials.

**3.Design**

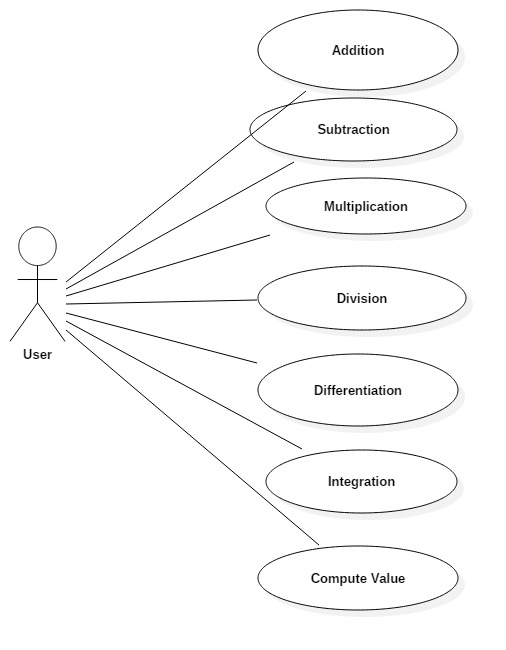
**3.1. Relational Diagram**

For implementing this application I’ve chosen an approach as close as possible to the ideal Object Oriented Paradigm, in the sense that each class is given one task only, and also, by the concept of inheritance code can be reused avoiding duplication.

From the diagram it can be easily seen, besides the structure of each class, the relationships among them, which will define as a whole, the project. As it can be observed, four main relationships appear most frequently: generalization, dependency, aggregation and composition.

Perhaps the first relation that can be seen is the one between classes **CoeffDegree** and **Polynomial**, since those are the basic classes on which the application is constructed. In this case, we would talk about composition, because an object of type **Polynomial**  would not exist unless the class **CoeffDegree** is present. Based on the same principle, composition can be found also in the gui classes, **LowerPanel**, **UpperPanel** and **MainWindow**.

Another aspect of Object Oriented Programming is shown by the generalization relationship, that appear in the design of operation classes. It can be seen that there exists a super-class, **Operation**, from where 3 other sub-classes derive: **BinaryOperation, UnaryOperation** and **ComputeValue**. In this manner, from **BinaryOperation** classes designated to arithmetic operations derived, and from **UnaryOperation**, differentiation and integration respectively.

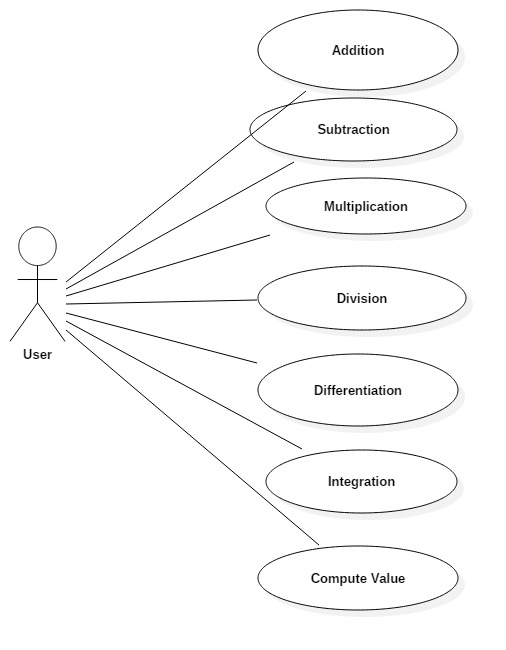
Moreover, dependency is present between **Operation** and **Polynomial** classes, whereas aggregation links **Operation**, **BtnsHandler** and **Polynomial**.

**Use Case Diagram**

From the previous diagram, it can be observed that the user can choose from a set of seven operations implemented on polynomials. As they wish, by using the user interface of the program, they could choose from the arithmetic operations addition, subtraction, multiplication or division to even compute the value in a given value, provided from the input.

**3.2 Classes design**

Further on, the most important attributes and methods of every class will be presented together with a brief description of every class. This is a good way to notice some similarities that appear in the code and in the applied idea. Moreover, we can observe the some relationships that appear among them, like inheritance for instance.

1. **CoeffDegree**

Maybe the most important class, it is the one on which class Polynomial is constructed. It has 2 private fields of type int( **coefficient, degree** ), and a field of type Boolean “used”, which is a helper when implementing the operations because it signals if a certain pair of “**CoeffDegree**” has been taken into consideration or not.

This class, among the getters and setters for its private fields, contain a constructor with 2 parameters which will initialize the fields “coefficient” and “degree” (**public CoeffDegree(int coeeficient, int degree)**). “**Used**” does not need a special initialization because it is first set to false for each pair. Moreover, the class implements one interface, more exactly “Comparable”, since it needs to have defined a certain order in which the objects of this type will appear in the polynomial. In order to do so, the method “**compareTo**” is overridden.

1. **Polynomial**

Polynomial class contains a single field and that would be called “coeffDegree”. This is actually an ArrayList containing objects of type “CoeffDegree”, as imposed through generics.

The parameter-less constructor has the job only to instantiate the list. As methods, there are the getters and setters which allow accessing the list from outside the class, and two overloaded methods called “insertValues”. One of them would receive an object of type “CoeffDegree”, whereas the other one, two ints. The result for each is the same, adding a new element in the list.

1. **Operation**

It is the abstract supperclass of all implemented operations. It cannot be instantiated, however it is needed for the sake of polymorphism, and to provide two static methods.

**ShowPolynomial** - is given a Polynomial as parameter and returns a string used to display it formatted;

**removeNullPairs** – has a parameter of type Polynomial, and returns another one, where all the pairs with zero coefficients are removed.

It also has two children: “UnaryOperation” and BinaryOperation”.

1. **UnaryOperation – abstract**

It is extending Operation class and has an abstract method called “doOpperation”; it gets one Polynomial type parameter and returns another Polynomial.

1. **Differentiation**

Extends “UnaryOperation” and implements “doOpperation”, returning the differentiated polynomial;

Contains the default constructor only.

1. **Integration**

Extends “UnaryOperation” and implements “doOpperation”. Does not have any fields or a different constructor, other than the default one.

1. **BinaryOperation - abstract**

Extends Operation. It is the superclass of the classes that provide solutions for the arithmetic operations performed on polynomials. Moreover, it contains, as UnaryOperation, a header of “doOpperation” method. This time, the method gets two parameters of type Polynomial. As opposed to the other similar class, BinaryOperation provides the method “insertRemains”, which will take two parameters and will decide whether or not a certain pairs has to be copied in the result polynomial. This superclass has 4 children, each one representing an arithmetic operation.

1. **Addition**

Extends BinaryOperation and implements one single method, doOperation. Trough this, it provides a solution to adding the polynomials.

1. **Subtraction**

Extends BinaryOperation and implements doOpperation. In order to compute the subtraction of the polynomials, it needs an additional method, similar with “insertRemains’ from the superclass, which will also change the sign of the second polynomials unused monomes.

1. **Multiplication**

Extends BinaryOperation and provides a solution to multiplying two polynomials. It also has another method called “touchUpPolnomial”, used to compute the sum of each CoeffDegree pair.

1. **Division**

It is the last derived class of BinaryOperation and provides a computation for dividing two polynomials. As fields ,it contains two Polynomials (quotient and remainder), where the results of “doOpperation” will be stored. Besides the getters and setters, there is a need for a method to return the leading degree and the coefficient respectively, so, Division implements these two methods named: getLeadingDegree for degree, and getLeadingCoefficient, for coefficient.

As it can be observed, the operations classes have no constructors and in principle, no field, except the division class, but methods which implement the desire operations.

The application does also come with a user interface in order to make user’s job easier. For the sake of Object Oriented Programming paradigm, the interface is done in a manner which is in concordance with the style of programming.

1. **UpperPanel extends JPanel**

As the class it’s extending, we can observe that it is actually a panel, which contains 4 private fields used for displaying the results of operations done over the polynomials:

* Private JTextField polynomialInput – the input string which is going to be interpreted as a polynomial
* Private JLabel display – output string of the result
* Private GridLayout upperPanLayout – layout of the panel is designed as a grid of two lines and 1 culumn
* Private Font font – manages the layout of the text displayed

**UpperPanel** has just one constructor **public UpperPanel(),**  with no parameters, which sets the background color, layout, and adds the two components to the panel. Nevertheless, it does contain two methods:

* **private JLabel editDisplay (JLabel display) -** sets the font for “**display**”
* **private JTextField editTextField (JTextField text) –** edit properties of the text field object

1. **LowerPanel extends JPanel**

Again, **LowerPanel**, similarly to **UpperPanel**, is an object of type panel, since it still extends class **Jpanel**.

It has 10 instances of objects whith suggestive names which will handle the operations:

* **private JButton add0;**
* **private JButton subtract;**
* **private JButton multiply;**
* **private JButton diferentiate;**
* **private JButton integrate;**
* **private JButton divide;**
* **private JButton clear;**
* **private JButton equal;**
* **private GridLayout lowerPanLayout = new GridLayout(2, 4);**
* **private Font btnFont;**

Moreover, it has two constructors, one parameterless, and the other one having a parameter of type BtnsHandler:

* **public LowerPanel() –** sets all the components and link them to the panel
* **public LowerPanel(BtnsHandler handler) –** call the parameterless constructor and the method **add Handler (BtnsHandler handler)** in order to, as the name says, allocate an object which will handle the events accordingly for each button;

Three additional methods are used:

* **public void addBtns()** – add all buttons to panel
* **public void addHandler(BtnsHandler handler) –** link the handler to each button
* **public void designButtons()**  - sets the font to each button

1. **OpperationLabel**

**OpperationLabel**  is a special class, since it is of type **enum ,** and its only scope is to keep constants, used for identifying what action will be performed when pressing different buttons : **ADD, SUBBSTRACT, MULTIPLY, DIVIDE, INTEGRATE, DIFFERETIATE, CLEAR, EQUAL**.

1. **MainWindow extends JFrame**

It is actually a Frame, since JFrame is the super-class, and has 5 attributes:

* **Private Dimension framesize –** sets the size of the frame to 800 over 400
* **Private JPanel upperPanel** – instance of on object of type UpperPanel
* **Private BtnsHandler handler** – instance of an object that will take care of the events triggered by pressing different buttons
* **Private JPanel lowerPanel** – instance of type LowerPanel
* **Private GridLayout frameLayout**  - as its name says, it sets the layout of the panel to two lines and one column

Only one constructor is defined here, **public MainWindow()**, which has the job on initializing the properties of this class.

1. **BtnsHandler implements ActionListener**

This is the crucial class the links the logic part of the application to its graphical interface. As we can see, BtnsHandler is the one to decide which operation is to be performed in case one button is pressed. As attributes, it contains 4 instances and this would be:

* **Private JPanel displayPanel** – needed for displaying the result
* **Private ArrayList<Integer> inputPol**  - list of integers, in the form of an ArrayList, which is going to keep in memory the coefficients of the polynomials inserted by the user, which will further be computed by the logic of the program
* **Private Polynomial pol1**  - first polynomial: only this one will be used by unary operations, like integration and differentiation
* **Private Polynomial pol2 ­**– second object of type Polynomial

**BtnsHandler** uses two constructors:

* **Public BtnsHandler()** – instantiate each object
* **Public BtnsHandler(JPanel displayPaanel)** – calls the parameter less constructor and also instantiate the display panel

Since it implements the interface **ActionListener**, the first method to be implemented has to be **ActionPerformed(ActionEvent btn).** This one is maybe the most important one from the whole class, since it is the method through which the systems knows how to handle the events brought by the buttons. In principle, what it does is first to decide which is the cause of the event, and then instantiate “operation” with the correct type such that “**doOperation()**” method will compute the desired result.

Nevertheless, **BtnsHandler** does contain three more methods that help the application reach is final scope:

* **private void breakString(String input)**  - uses StringTokenizer in order to correctly create a polynomial object. This will break an input string in tokens, separated by the characters “x” and “^”
* **private Polynomial createPolynomial()** – creates the polynomial based on the strings computed by breakStrings() method
* **private void clearPolynomials()** – empties the list of coefficients for the two polynomials

Moreover, the application provides some test classes for each operation. Each one has a predefined Polynomial that contains the result that is expected, a constructor that initializes its values and a  **test()**  method, returning the correctness true a Boolean value.

* **AdditionTest**
* **DifferentiateTest**
* **DivisionTest**
* **IntegrateTest**
* **MultiplicationTest**
* **OperationTest**
* **SubtractionTest**

1. **StartApplication**

This is the main class, containing one method **public static void main(String [ ] args)**. This method initiates the launch of the application cration a new **MainWindow** object.

**3.3.Packages**

Each set of entities are grouped in a different package, in order to distinguish between different parts of the program.

1. polynomial\_op.setup: - StartApplication
2. polynomial\_op.operations:
   * 1. Addition
     2. BinaryOperation
     3. Differentiation
     4. Division
     5. Integration
     6. Multiplication
     7. Operation
     8. Subtraction
     9. UnaryOperation
3. polynomial\_op.gui:
   * 1. BtnsHandler
     2. LowerPanel
     3. MainWindow
     4. OpperationLabel
     5. UpperPanel
4. polynomial\_op.entities:
   * 1. CoeffDegree
     2. Polynomial
5. Polynomial\_op.tests:
   * 1. AdditionTest
     2. DifferentiateTest
     3. DivisionTest
     4. IntegrateTest
     5. MultiplicationTest
     6. OperationTest
     7. SubtractionTest

**4.Results**

The results obtained in the level of testing the application confirm the fact that it reaches its objective: it implements the operations performed giving the exact results becoming an interesting and interactive tool for education and, why no, a mathematical utility for processing polynomials in the engineering field. The complexity of the system is not high in the actual level of development, but by developing the program could become an etalon for making operations on polynomials.

**5.Conclusions**

Polynomials represent an interesting subject in the mathematical field by which we can describe events or states for the engineering field and not only. The operations with a high degree are difficult if they are performed by a human being, that is why the integration in the frame of an application is more than necessary.

The application for processing operations on polynomials is an efficient, simple and user friendly program. It performs basic operations (like addition, subtraction, multiplication, division) but also more advanced operations such as integration, differentiation.

The Graphical user interface is user friendly and responsive to the commands executed by the user, the graphical elements are very well defined and they give no ambiguity to the user.

By mentioning that the application is in its final state of development, now, it presents some drawbacks. The first thing I want to emphasis is scalability. If at the moment it processes polynomials with integer coefficients it, further developments would mean to extend the domain to BigInteger. Also, another possible improvement for a future extension is to process polynomials that have more than one variable.

Personally, realizing this application helped me by reminding me the concepts of Objective Oriented Programming studied last semester and also the usage of the Swing library.

The application could be improved by performing some other operations like n order derivative or integration, getting the polynomial at a certain power, finding the roots of a certain polynomial.

**6.References**

*[1]* [*www.stackoverflow.com*](http://www.stackoverflow.com)

*[2]* *docs.oracle.com/javase/7/docs/*