

## Universitatea Tehnică Cluj-Napoca

## Facultatea de Automatică şi Calculatoare

## Secţia: Calculatoare, engleză

## Programming Techniques

## ~ Polynomials ~

Student: Ban Bogdan Marian

Group id: 30425, second year

Professor: Salomie Ioan

Table of Contents

1. Task Objectives------------------------------------------------------------03

2. Problem analysis, modelling, scenarios, use cases--------------------03

3. Projection (UML diagrams, data structures, class projections, -----07

Interface, relations, packages, algorithms,)

4. Implementation and testing-----------------------------------------------15

5. Results-----------------------------------------------------------------------16

6. Conclusions-----------------------------------------------------------------16

7. Bibliography----------------------------------------------------------------16

1. Task Objectives

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

2. Problem analysis, modelling, scenarios, use cases

Problem analysis

The problem required was to read polynomials from keyboard by a user and calculate possible operations between the two polynomials. In order to implement this application one requires knowledge from math but also from different algorithms.

In mathematics, a polynomial is an expression consisting of variables and coefficients which only employs the operations of addition, subtraction, multiplication, and non-negative integer exponents. An example of a polynomial of a single variable x is x^2 + 4x + 7.

A polynomial is an expression that can be built from constants and symbols called indeterminates or variables by means of addition, multiplication and exponentiation to a non-negative power. Two such expressions that may be transformed, one to the other, by applying the usual properties of commutativity, associativity and distributivity of addition and multiplication are considered as defining the same polynomial.

A polynomial in a single indeterminate x can always be written (or rewritten) in the form:



For example:

 -5x^2y 

is a term. The coefficient is −5, the indeterminates are *x* and *y*, the degree of *x* is two, while the degree of *y* is one. The degree of the entire term is the sum of the degrees of each indeterminate in it, so in this example the degree is 2 + 1 = 3.

Forming a sum of several terms produces a polynomial. For example, the following is a polynomial:

\underbrace{_\,3x^2}_{\begin{smallmatrix}\mathrm{term}\\\mathrm{1}\end{smallmatrix}} \underbrace{-_\,5x}_{\begin{smallmatrix}\mathrm{term}\\\mathrm{2}\end{smallmatrix}} \underbrace{+_\,4}_{\begin{smallmatrix}\mathrm{term}\\\mathrm{3}\end{smallmatrix}}. 

It consists of three terms: the first is degree two, the second is degree one, and the third is degree zero.

Polynomials of small degree have been given specific names. A polynomial of degree zero is a constant polynomial or simply a constant.

Polynomials of degree one, two or three are respectively linear polynomials, quadratic polynomials and cubic polynomials. For higher degrees the specific names are not commonly used, although quartic polynomial (for degree four) and quintic polynomial (for degree five) are sometimes used. The names for the degrees may be applied to the polynomial or to its terms. For example, in x^2 + 2x + 1 the term 2x is a linear term in a quadratic polynomial.

Add and multiplication

We define on R[X] two algebraic operations: adding and multiplying.

Properties:

* Commutative f+g=g+f
* Associative (f+g)h=f(g+h)
* There is a neutral element for adding f+f0=f
* There exists an inverse element for any element f+(-f)=0

Add for polynomials:

Let ,  two elements from R[X], then we define:

, 

Properties of polynomials

Associativity

,  R[X]

It is true, if ,and  then we have so .

Similar, we obtain 

In conclusion , for any .

Commutativity

, R[X]

It is true, if  şi , we have,

In conclusion  for any . So .

Neutral element

Constant polynoamial 0=(0,0,0,…) is an element for adding polynomials, meaning that any R[X],we have:



Inverse elements

For any polynomial R[X], there is polynomial noted , so:



Multiplication for polynomials

Let , 

We define





Properties of polynomials

Associativity

For any R[X], we have:



Commutativity

For any R[X], we have:



It is true, if , , then noting  and, we have

 and .Since adding and multiplying are associative and commutative, we have cr=dr, for any . So .

Neutral element

Constant polynoamial 1=(1,0,0,…) is an element for multiplying polynomials, meaning that any R[X],we have: 

Inverse elements

R[X] is inversable if ,so.:



Distributivity

For any R[X], the following relation is true:



Modelling

To keep the project simple, I decided to create a Polynomial class which contains the possible operations. When building a new polynomial it is required a degree and an array of coefficients.

I also made two classes that create the user interface. Besides the main class (which only contains the main) I also created two listener classes for the application.

Scenarios and Use cases

The user can easily use the application, he/she just needs to introduce coefficients for a polynomial of type int. If the insertion is wrongly written , an error message will appear. Add, subtract, multiply , division , integrate , and derivate are the operations available for the user.

3.Projection

Class diagram:

**ReadKeyListener**

switcher: int

ReadKeyListener(Application,int)

keyPressed(KeyEvent):void

keyReleased(KeyEvent):void

keyTyped(KeyEvent):void

**MainPanel**

textPol1: JTextField

textPol2: JTextField

areaOutput: JTextArea

arealnput: JTextArea

add: JButton

subtract: JButton

multiplication: JButton

division: JButton

integrate: JButton

derivate: JButton

polLabel: JLabel

operationLabel: JLabel

infoLabel: JLabel

outputLabel: JLabel

inputLabel: JLabel

MIDDLE COORDONATE: int

MainPanel()

setBound(Component,Rectangle):void

getTextPoll ():JTextField

setTextPoll (JTextField):void

getTextPol2 ():JTextField

setTextPol2(JTextField):void

getArealnput():JTextArea

setArealnput(JTextArea):void

getAreaOutput(): JTextArea

setAreaOutput(TextArea): void

getInfoLabel() : JLabel

setInfoLabel(label):void

getAdd():JButton

getSubstract():JButton

getMultiplication: JButton

getdivision(): JButton getlntegrate()::JButton

getDerivate():JButton

**OperationListener**

OperationListener(Application)

ActionPerformed(ActionEvent):void

**Application**

setP1(Polynom):void

setP2(Polynom):void

Application()

readPolynom(JTextField):Polynom

reset():void

getP1():Polynom

getP2():Polynom

updateOutputText(String):void

getPanel():MainPanel

**Polynom**

degree: int

coef: int[]

Polynom(int,int[])

printString():String

addP(Polynom): Polynom

subtractP(Polynom):Polynom

multiplyP(Polynom):Polynom

derivateP():Polynom

integrateP():Polynom

divideP(Polynom):Polynom

**Main**

main(String[]):void

Class projection

The classes chosen for this project are specific and each one of them handles an important part of the program in order to create the application as easy, clean and fast as possible.

1. polynomials package

This package contains the Main and Polynom class.

Polynom Class: this class is the concept of a polynomial , it is constructed by stating the degree and the coefficients in a vector. It also has all the logic methods for arithmetic.

Main Class : this class contains the main, in which we create a new Application.

1. graphic package

This package contains the classes that are related to the graphical user interface.

Application Class : it a a JFrame class, in here we declare our polynomials and send them to other classes. This will create our window in which we put everything regarding the user interface.

MainPanel Class : it is a JPanel class, here we create the graphic itself , interacting with the user, with all its buttons, text fields , input windows, output windows.

1. listeners package

This package contains the classes which implement a certain listener

OperationListener Class : this class handles with the action performed by the buttons and showing the right output.

ReadKeyListener: this class also handles some events, whenever a key is released, the input window updates automatically

Algortihms

For creating and reading the polynomials I used the following algorithm:

**public** Polynom readPolynom(JTextField textField) {

**int** coef[] = **new** **int**[10];

**int** i = 0;

StringTokenizer c = **new** StringTokenizer(textField.getText().toLowerCase().trim());

**while** (c.hasMoreTokens()) {

coef[i] = Integer.*parseInt*(c.nextToken());

i++;

}

Polynom p = **new** Polynom(i - 1, coef);

**return** p;

}

We send as a parameter the type of input from which we extract the data and after that with the help of the StringTokenizer method we do the reading of the coefficients. They are separated with spaces when written in the input. One by one each coefficient is introduced in the vector coef[] which will create the final polynomial.

To make sure everything works smoothly, I defined an action listener for each button event or reading event in order for the application to make the correct choices.

**public** **void** actionPerformed(ActionEvent e) {

Polynom rez;

String whichButton = e.getActionCommand();

**if** (whichButton.equals("Add")) {

rez = application.getP1().addP(application.getP2());

application.updateOutputText("The adding result :\n" + rez.printString());

} **else** **if** (whichButton.equals("Subtract")) {

rez = application.getP1().subtractP(application.getP2());

application.updateOutputText("The subtracting result :\n" + rez.printString());

} **else** **if** (whichButton.equals("Multiply")) {

rez = application.getP1().multiplyP(application.getP2());

application.updateOutputText("The multiplying result :\n" + rez.printString());

} **else** **if** (whichButton.equals("Derivate")) {

rez = application.getP1().derivateP();

application.updateOutputText("The derivating result :\n" + rez.printString());

} **else** **if** (whichButton.equals("Integrate")) {

rez = application.getP1().integrateP();

application.updateOutputText("The integrating result :\n" + rez.printString());

} **else** **if** (whichButton.equals("Divide")) {

rez = application.getP1().divideP(application.getP2());

application.updateOutputText("The dividing result :\n" + rez.printString());

}

application.reset();

}

The algorithm for adding two polynomials is the next one:

**public** Polynom addP(Polynom p) {

**if** (**this**.degree > p.degree) {

rez = **new** Polynom(**this**.degree, **this**.coef);

**for** (**int** i = 0; i <= degree; i++) {

rez.coef[i] += p.coef[i];

}

**return** rez;

} **else** {

rez = **new** Polynom(p.degree, p.coef);

**for** (**int** i = 0; i <= degree; i++) {

rez.coef[i] += **this**.coef[i];

}

**return** rez;

}

}

First we make a comparison between the degrees of the polynomials that will be added in order to know which will be the degree of the final polynomial between the two.

Then we traverse with a cycle the polynomial with the smaller degree and add its coefficients to the polynomial with the higher degree. The result being recorded on a third polynomial.

The algorithm for subtracting two polynomials is the next one :

**public** Polynom subtractP(Polynom p) {

**int** ok = 1;

**if** (**this**.degree >= p.degree) {

rez = **new** Polynom(**this**.degree, **this**.coef);

**for** (**int** i = 0; i <= degree; i++) {

rez.coef[i] = rez.coef[i] - p.coef[i];

}

} **else** {

rez = **new** Polynom(p.degree, p.coef);

**for** (**int** i = 0; i <= degree; i++) {

rez.coef[i] = rez.coef[i] - **this**.coef[i];

}

}

**while** (ok == 1) {

**if** ((rez.coef[rez.degree] == 0) && (rez.degree > 0)) {

rez.degree = rez.degree - 1;

} **else** {

ok = 0;

}

}

**return** rez;

}

Almost the same like adding but we must verify if the degree of the polynomial becomes lower

The algorithm for multiplying two polynomials is the next one:

**public** Polynom multiplyP(Polynom p) {

**int** i, j, degreef;

**int** coeff[];

degreef = **this**.degree + p.degree;

coeff = **new** **int**[degreef + 1];

**for** (i = 0; i <= degreef; i++) {

coeff[i] = 0;

}

**for** (i = 0; i <= **this**.degree; i++) {

**for** (j = 0; j <= p.degree; j++) {

coeff[i + j] += **this**.coef[i] \* p.coef[j];

}

}

rez = **new** Polynom(degreef, coeff);

**return** rez;

}

Starting from constructing a vector initialized with 0, this will contain the coefficients of the final polynomial. The degree of this polynomial will be equal to the sum of the degrees we multiply. We traverse with two cycles for the coefficients of the two polynomials in order to obtain the final one.

The algorithm for derivating a polynomial is the next one:

**public** Polynom derivateP() {

**int** i;

rez = **new** Polynom(**this**.degree - 1, **this**.coef);

**for** (i = 1; i <= **this**.degree; i++) {

rez.coef[i - 1] = **this**.coef[i] \* i;

}

**return** rez;

}

We cycle with a for loop so that the coefficient from the position i of the polynomial to take the value from the position i+1 to calculate the derivate, times the value of i

The algorithm for integrating a polynomial is the next one:

**public** Polynom integrateP() {

**int** i;

rez = **new** Polynom(**this**.degree + 1, **this**.coef);

**for** (i = 0; i <= **this**.degree; i++) {

rez.coef[i + 1] = **this**.coef[i] / (i + 1);

}

rez.coef[0] = 0;

**return** rez;

}

The hardest algorithm to implement was the dividing of two polynomials.

The algorithm for it is the next one:

**public** Polynom divideP(Polynom p) {

Polynom p1 = **this**;

**int** i;

**int**[] newCoef = **new** **int**[10];

**if** ((p.degree == 0) && (p.coef[0] == 0)) {

**throw** **new** RuntimeException("Divide by zero polynomial");

}

**if** (p1.degree <= p.degree) {

**return** **new** Polynom(0, coef);

}

**for** (i = 0; i <= **this**.degree; i++) {

newCoef[i] = p1.coef[p1.degree] / (p.coef[p.degree]);

}

**int** exponent = p1.degree - p.degree;

Polynom p2 = **new** Polynom(exponent, newCoef);

**return** p2.addP(p1.subtractP(p.multiplyP(p2)).divideP(p));

}

It has three possible outcomes. If the degree of the coefficient we divide is 0 and its coefficients are 0 it throws and exception “Divide by zero polynomial”.

If the degree of the polynomial we divide is greater than the one it is divided we return a new polynomial with degree 0

Else, we return a recurse method for the division of the two polynomials.

We cycle with a for loop the coefficient vector to calculate the integration.

The algorithm for live input is the next one:

**public** **void** keyReleased(KeyEvent arg0) {

**if** (switcher == 1) {

**try** {

application.setP1(application.readPolynom((JTextField) arg0.getSource()));

} **catch** (Exception ex) {

application.getPanel().getInfoLabel().setText("Error! Please check the input");

}

**if** (application.getP1() != **null**) {

**if** (application.getP2() != **null**) {

application.getPanel().getAreaInput()

.setText(application.getP1().printString() + "\n" + application.getP2().printString());

} **else** {

application.getPanel().getAreaInput().setText(application.getP1().printString());

}

}

}

**if** (switcher == 2) {

**try** {

application.setP2(application.readPolynom((JTextField) arg0.getSource()));

} **catch** (Exception ex) {

application.getPanel().getInfoLabel().setText("Error! Please check the input");

}

**if** (application.getP2() != **null**) {

**if** (application.getP1() != **null**) {

application.getPanel().getAreaInput()

.setText(application.getP1().printString() + "\n" + application.getP2().printString());

} **else** {

application.getPanel().getAreaInput().setText(application.getP2().printString());

}

}

}

}

From the overridden method KeyReleased() we have two cases. If switcher is 1 we set the polynomial for the first Text Field , otherwise if the switcher is 2 we set the polynomial for the second Text Field and we print the string in the input window.

Overall each algorithm does a specific thing and that makes the program with low coupling.

I did not see any more necessity to add more classes the module it even more because I think I already refactored it enough to have a certain standard.

User Interface

It is created by the Application class which has added a MainPanel object with all its components. The interface is friendly for the user having instructions at the bottom of the application of how to use it. Moreover it has tool tips for certain buttons.

The input commands can be live easily observed as well as the output after performing a certain operation.

I created using rectangle for each element in the panel.

I used this method to make an easier code to understand

**public** **void** setBound(Component comp, Rectangle bounds) {

comp.setBounds(bounds);

}

This is the part from declaring certain elements for the JLabel

**private** JTextField textPol1 = **new** JTextField(30);

**private** JTextField textPol2 = **new** JTextField(30);

**private** JTextArea areaOutput = **new** JTextArea();

**private** JTextArea areaInput = **new** JTextArea();

**private** JButton add = **new** JButton("Add");

**private** JButton subtract = **new** JButton("Subtract");

**private** JButton multiplication = **new** JButton("Multiply");

**private** JButton division = **new** JButton("Divide");

**private** JButton integrate = **new** JButton("Integrate");

**private** JButton derivate = **new** JButton("Derivate");

**private** JLabel polLabel = **new** JLabel("Enter the polynomials:");

**private** JLabel operationLabel = **new** JLabel("Choose the operation:");

**private** JLabel infoLabel = **new** JLabel("Write the coefficients of the polynomials with space between them");

**private** JLabel outputLabel = **new** JLabel("Your output:");

**private** JLabel inputLabel = **new** JLabel("Your input:");

**private** **static** **final** **int** ***MIDDLE\_COORDONATE*** = 80;

Use Case diagram

This diagram shows the actor (user) and what he can perform using the application.

Sequence diagram:

Application

MainPanel

P1

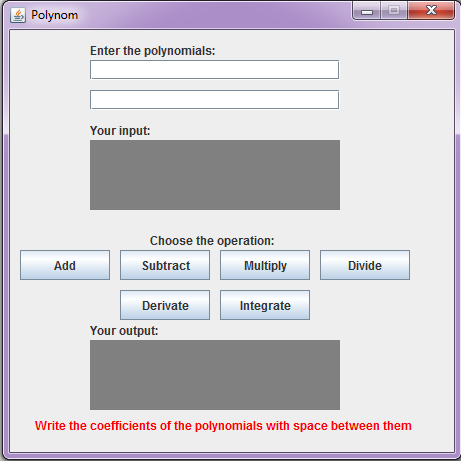
P2

Polynom

As seen from the class diagram, the Application creates the two polynomials which are sent in the operations from the Polynom class and then the output is shown in the MainPanel.

4. Implementation and testing

At first, I implemented the user interface because that is how I prefer , knowing which buttons and how many text areas or fields I will have to use. After that I created the concept of the polynomial and tested the input reading from it and showing the input. After that I started with the arithmetic.

Testing:

This is how the interface looks like. I believe it looks clear and easy to use.

In order to properly use it , the user must type in the coefficients of both the polynomials with space between them. If they type something different than number, an error will display at the bottom. After writing the input they will notice it in the first window and after applying the operation the output will show the result of the desired operation.

5. Results

I developed an easy application to use, that proves to be useful when one wants to perform simple arithmetic for two polynomials.

6. Conclusions

I am satisfied with how the project ended up to be, it sharped my skills of OOP from the first semester and I will definitely continue to grow in java programming language because I enjoy it a lot.

Of course there are always way to improve and application, for example for this one I can implement the graph of the polynomials, the roots of the polynomials, adding 3 polynomials and performing the operations for them or other improvements.

7. Bibliography

<https://en.wikipedia.org/wiki/Polynomial>

<http://www.loufranco.com/wp-content/uploads/2012/11/cheatsheet.pdf>