**Polynomial Calculator**

**1.Main Objective**

The main purpose of this homework is work with one of the most common structures in Computer Science, named Polynomials and its algebraic operations. The advantage of using polynomials is that they can approximate derivable functions and the simplicity of their operations. So that, we will always find polynomials with a set of functions which can be manipulated with ease.

The main purpose of this project is to implement the algebraic functions of the polynomials which are addition, substraction, multiplication, division, integration and derivation.

**2.Problem Analysis and Modeling**

When we are taking about the analysis of a problem, we mainly talk about an abstract set of operations and properties with the help of which we are able to find some characteristics of the processes. This advantage is done by the object oriented programming; this strategy is called bottom-up design.

Usually we start from the specifications of the project, where we are looking for:

* Noun – which becomes potential classes;
* Verbs- which can play the role of the methods inside the classes.

As the application has as its main purpose the portability between the users, the interface is the one which will make the connection between the user and the program. In our case, the user will introduce a canonic form of a polynomial as a string.

The use case is the description of a set of sequences of actions, including variants that a system performs to yield an observable result of value to an actor

The actor has a set of roles that users of use cases play when interacting with these use cases. Typically, an actor represents a role that a human or hardware device or even another system plays with a system.

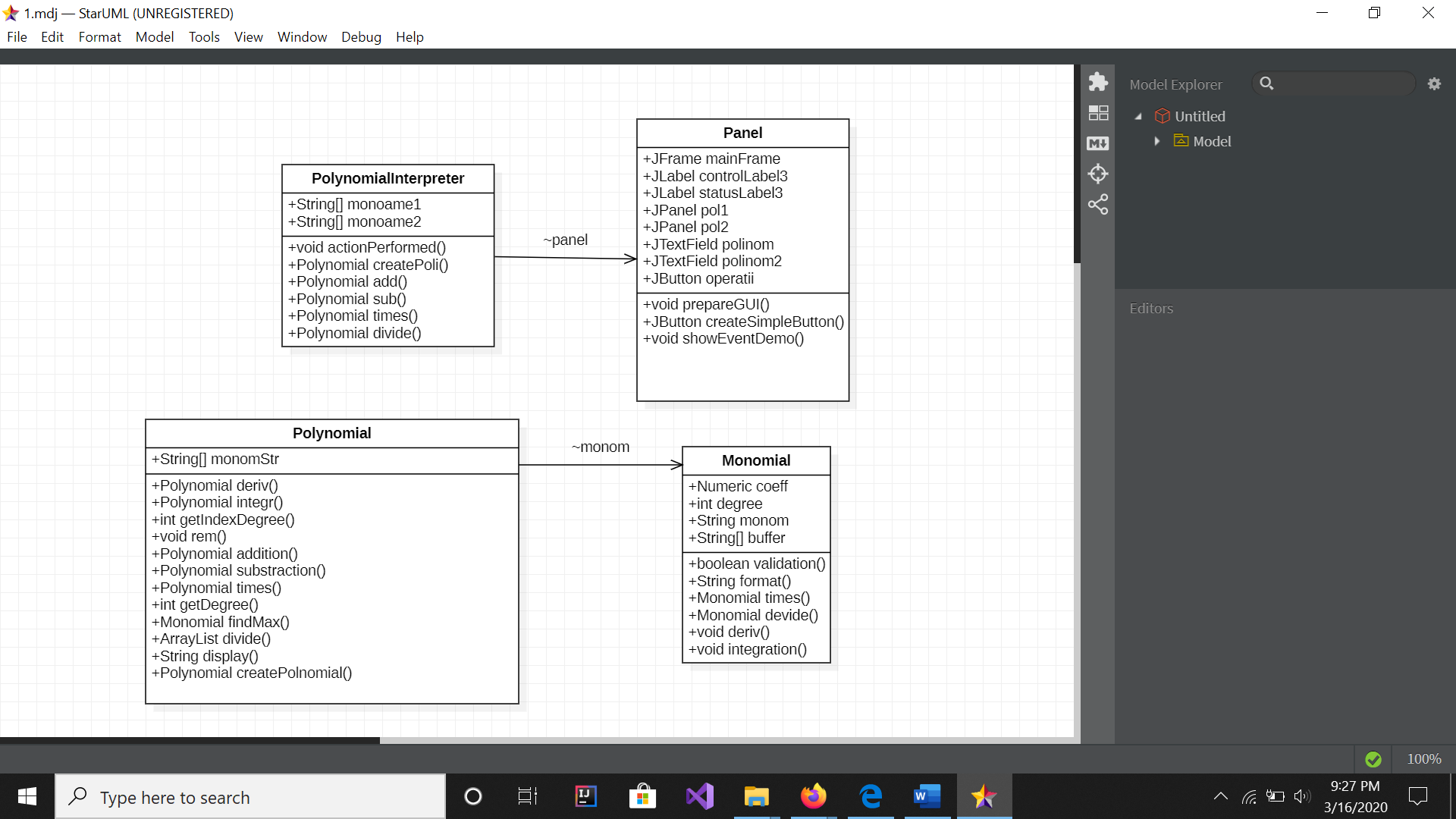
**2.1 Utilization cases**

Once we have established the main classes and methods of our application we should take into consideration the limit cases that can appear:

* To calculate the sum/ difference/ multiplication/ division between the first and second polynomial.
* To calculate the integration and derivation of both polynomials.
* Special words used in the strings which are converted into polynomials.

The usual utilization of the program refers to introduce the first polynomial in its classic format. The process is the same also for the second polynomial, after which we will be able to perform the elementary operations which are presented each one on a button, being recognized with its specific symbol. Even though, sometimes unusual things can happen, such as introducing a text without a numerical significance or introducing one additional unknown besides the establish one.

**3. Projection**



The polynomial calculator has 4 classes and a main class which instantiate them.

1.Panel – with its help we will create the user interface which will be used in the whole project. Also, here we will instantiate the listeners which have the role of capturing the possible actions of the user using the events. So that, we will have different listeners for buttons and text fields.

2. PolynomialInterpreter – this class processes the data which is caught by the listeners and activates the needed methods.

3.Polynomial – it enables us to create the polynomials as a list of monomials, which are fundamental entities. So that, we take into consideration the case when the user does not introduce correctly a polynomial.

4. Monomial – it is a fundamental class for the project due to the fact that it defines the monomial structure, an entity which contains a coefficient and a degree. Also, here we will define the main operations: addition, substraction, integration, derivation, multiplication and division.

We have created 2 packages: one called Poli where we have 2 classes ( monomial and polynomial) which enables us to construct a polynomial put of a mononomial and a package called Controller which lets us create the user interface using the swing Java Class.

In this application we can also find a predefined algorithm which enables us to extract digits from a string.

int i=0;  
while(i<buffer[0].length())  
{  
 char letter= buffer[0].charAt(i);  
 if(letter=='-') nrStr+=letter;  
 if(letter>='0' && letter<='9') nrStr+=letter;

**4. Implementation**

**4.1 Panel**

Class Panel is made using Swing API which Is part of the Java Foundation Classes (JFC). It offers facilities to write applications with a graphical user interface and also includes 17 packages consisting of classes and interfaces.

The most important package from Swing is javax.swing which contains the basic components to create graphical user interface, atomic components (e.g. JLabel, JButton, JCheckBox ), complex components (e.g. JTable), components for editing text (e.g. JTextField) , menus, intermediate containers (e.g.JPanel, JTabbedPane, JDesktopPane) and high level containers (e.g. JFrame, JDialog, JApplet ).

JFrame is a high level container that allows to display graphical components on the screen.

mainFrame = new JFrame("Poli.Polynomial Calculator");  
mainFrame.setSize(700,200);

The default behaviour of a JFrame object when we close the window is to hide the window when the user presses the close button.

jf.setDefaultCloseOperation (WindowConstants.HIDE\_ON\_CLOSE);

jf.setDefaultCloseOperation (WindowConstants.DO\_NOTHING\_ON\_CLOSE);

jf.setDefaultCloseOperation (JFrame.EXIT\_ON\_CLOSE);

In this class is also used Layout Mnagers which are object that controls the size and position of the components within a container which lets us implement the LayoutManeger Inteface.

The Layout Manager we have used are :

BorderLayout - divides the display area in five regions: north, south, east, west, and center; when you add a new component, you need to specify the position.

pol2.setBorder(BorderFactory.*createLineBorder*(Color.*BLACK*));

GridLayout - places components in a grid of cells (the number of rows and columns are fixed); each component takes all the available space within its cell, and each cell is exactly the same size.

mainFrame.setLayout(new GridLayout(1,1));

Here, we have also implemented the method showEventDemo() which has the listeners, which will be further used in the PolynomialInterpreter.

deriv.setActionCommand("deriv");  
// instanta obiect cate PolynomialInterpreter  
plus.addActionListener(new PolynomialInterpreter(this));

**4.2. PolynomialInterpreter**

Is the class which takes into consideration all the actions from the user and control them with some functionalities; this is why it implements the interface ActionListener and also the method actionPerformed().

One of the most important characteristics of this class is that is always contains 2 strings. The user will always introduce the information as a text in the JTextField. The information will not change as long as the user does not take action upon the text fields.

It does not matter the type of the action, at each new call we have to reinstantiate the polynomials, but the way they are created is the same. The method createPoli() is the first method which will make the first parsing of the strings.

At the end, this method will give to the constructor from the Polynomial class an array made out of strings, which will be again parsed.

String actualBuffer = (buffer.contains("-")) ? buffer.replace("-", "+-") : buffer;  
finalBuffer = (actualBuffer.charAt(0) == '+') ? actualBuffer.substring(1) : actualBuffer;  
monoame = finalBuffer.split("\\+");

**4.3. Polynomial**

The polynomial class is the one which enables us to really create the polynomials. It has only one public constructor and 2 private constructors which are used only inside this class.

In this class are created the algebraic operations which can be performed between two polynomials:

*4.3.1. The addition*

The addition of two polynomials is mostly based on the sum of the monomial’s coefficients as long as the degree of the monomial is the same. But, as we want the addition to be made on a general case, we have to take into consideration the unfavorable case, so that we can divide the addition in three cases:

1. Degree p1.monom < ∀ degree p2.monom => add p2.monom
2. Degree p1.monom = ∀ degree p2.monom => p1.monom + p2.monom and delete p2.monom
3. If ∃ degree p2.monom => add what it left

Polynomial addition(Polynomial P2)  
{  
 Polynomial P3 = new Polynomial();  
 int i=0;  
 while(i<this.monom.size())  
 {  
 int d1 = this.monom.get(i).getDegree();  
 Number c1 = this.monom.get(i).getCoeff();  
  
 int i\_degree = P2.getIndexDegree(d1);  
 if(i\_degree == -1)  
 {  
 //haven't been found => add new monomial  
 P3.monom.add(new Monomial(c1, d1));  
 }else{  
 //foud => eff the sum  
 int d3 = P2.monom.get(i\_degree).getDegree();  
 Number c3 = P2.monom.get(i\_degree).getCoeff();  
 P3.monom.add(new Monomial(c1.intValue()+c3.intValue(), d3));  
 //get rid of what we do have  
 P2.monom.remove(i\_degree);  
 }i++;  
 }  
 //add what we also have in p2  
 int j=0;  
 while(j<P2.monom.size())  
 {  
 int d2 = P2.monom.get(j).getDegree();  
 Number c2 = P2.monom.get(j).getCoeff();  
 P3.monom.add(new Monomial(c2,d2));  
 P3.rem();  
 j++;  
 }return P3; }

*4.3.2. The substraction*

The substraction is only a particular case of the addition where all the coefficients of the second polynomial are multiplied with -1.

1. (-1) \* p2.monom.coefficient
2. add(p1, p2)

*4.3.3. The multiplication*

public Polynomial times(Polynomial P2)  
{  
 Polynomial P3 = new Polynomial();  
 int i=0;  
 while(i<this.monom.size())  
 {  
 Monomial m1 = this.monom.get(i);  
 int j=0;  
 while(j<P2.monom.size())  
 {  
 Monomial m2= P2.monom.get(j);  
 Monomial m3 = m1.times(m2);  
 P3.monom.add(m3);  
 j++;  
 }i++;  
 }  
 return P3;  
}

*4.3.4. The integration*

int i=0;  
while(i<this.monom.size())  
{  
 this.monom.get(i).integration();  
 i++;

*4.3.5 The derivation*

public Polynomial deriv()  
{  
 int i=0;  
 while(i<this.monom.size())  
 {  
 this.monom.get(i).deriv();  
 i++;  
 }  
 return null;  
}

*4.3.6. The division*

public ArrayList<Polynomial>divide(Polynomial impartitor)  
{  
 ArrayList<Polynomial> p = new ArrayList<Polynomial>();  
 Polynomial cat = new Polynomial();  
 Polynomial rest = new Polynomial();  
 rest = this;  
 if(rest.getDegree() < impartitor.getDegree())  
 {  
 cat.monom.add(new Monomial(0,0));  
 }else{  
 while(rest.getDegree() >= impartitor.getDegree() /\*&& rest.value()!=0\*/)  
 {  
 Monomial dm = findMax(rest);  
 Monomial im = findMax(impartitor);  
 Monomial m = dm.devide(im);  
 Polynomial pm= new Polynomial(m);  
 cat.monom.add(m);  
 rest = rest.substraction(impartitor.times(pm));  
 }  
 }  
 if(rest.monom.isEmpty())  
 rest.monom.add(new Monomial(0,0));  
 p.add(cat);  
 p.add(rest);  
 return p;  
}

**4.4. Monomial**

In this class we have implemented the basic function for further implementing the algebraic operations in the class Polynomial.

But in this class, we can remark the necessity for a method of validation, which actually is a decisional tree for all the data which can be introduced by the user.

private boolean validation (String monom)  
{  
 //we should verify if there are any other symbol besides characters, sign and ^  
 if(!monom.matches("^[a-zA-Z0-9\\^\\\*\\-]\*")) return false;  
 Pattern polynomFormat = Pattern.*compile*("\\^");  
 Matcher match = polynomFormat.matcher(monom);  
 String s = new String();  
 while(match.find())  
 {  
 s=match.group();  
 }  
 if(s.isEmpty()) {  
 //if we do not have the symbol ^  
 buffer=monom.split("[a-zA-Z]");  
 if (buffer.length == 0) {  
 coeff=1;  
 degree=1;  
 } else {  
 coeff=(!buffer[0].isEmpty()) ? Integer.*parseInt*(buffer[0]) : 1;  
 degree=(buffer[0] == monom) ? 0 : 1;  
 } }  
 else  
 {  
 // has the symbol ^  
 buffer = monom.split("\\^");  
 try{  
 String nrStr = new String();  
 int i=0;  
 while(i<buffer[0].length())  
 {  
 char letter= buffer[0].charAt(i);  
 if(letter=='-') nrStr+=letter;  
 if(letter>='0' && letter<='9') nrStr+=letter;  
 i++;  
 }  
 coeff= (nrStr.isEmpty()) ?1:Integer.*parseInt*(nrStr);  
 degree=Integer.*parseInt*((buffer[1]));  
 }catch (NumberFormatException e)  
 {  
 System.*out*.println("Not a valid format");  
 }  
 }  
 return true;  
}

In a brief, we can conclude that the function defined above verify the string before creating the specific monomials:

Firstly, we verify if the string contains some special characters which should not be in the structure of the monomial such as ?, !, $. @. After having this validation, we are going to look after the specific symbol “^” and treat the situation which can appear : if we find it or not.

In the end, we will look after the coefficients of the unknown variable and analyze the cases when there is nothing in front of the unknown variable or when we do not have such a variable.

**5.Results**

In order to verify the correctitude of the functions implemented in our application, we are going to take some specific cases and make a test for each of them using software testing.

Software testing is the process of executing a piece of software/system to identify any gaps, errors, or missing requirements in contrary to the actual requirement. Types of testing : Unit testing and Integration testing.

We have implanted the unit test; The unit test is apiece of code written by a developer that executes a specific functionality in the code to be tested and asserts a certain behavior or state.

In order to make those tests we need to use the annotation @Test and the public class Assert which extends java.lang.Object.

@Test  
public void testAddition()  
{  
 Polynomial p\_expected = Polynomial.*createPolinomial*("10x^2+10x+3");  
 Polynomial p1 = Polynomial.*createPolinomial*("7x+3");  
 Polynomial p2 =Polynomial.*createPolinomial*("10x^2+3x");  
  
 *assertNotSame*(p\_expected,p1.addition(p2));  
}

**6. Conclusions**

To sum up my presentation of the application I want to remind the importance of the string parsing, and if I were to make my application more efficient I would make an additional class, maybe called “ Parsing” which will give me the possibility not to always have something to link the process of monomial determination to the creation of the polynomial.

As further development, we could add a dummy variable x, which will calculate the value of our polynomial ( e.g 8x+1 for x = 2, will have the value 17 ), or to create a new class “Cartesian” which will enable us to draw the polynomials on the axis.

**7. Bibliography**

[**https://stackoverflow.com/questions/4030928/extract-digits-from-a-string-in-java**](https://stackoverflow.com/questions/4030928/extract-digits-from-a-string-in-java)

[**https://mkyong.com/tutorials/junit-tutorials/**](https://mkyong.com/tutorials/junit-tutorials/)

[**http://users.utcluj.ro/~crisb\_pop/PTA1-SUPPORT-PRESENTATION.pdf**](http://users.utcluj.ro/~crisb_pop/PTA1-SUPPORT-PRESENTATION.pdf)

http://www.vogella.com/tutorials/JUnit/article.