PROGRAMMING TECHNIQUES

POLYNOMIAL CALCULATOR

STUDENT: ELEKES PÉTER
UNIVERSITY: UNIVERSITY OF TECHNICAL ENGINEERING CLUJ-NAPOCA
FACULTY: COMPUTER SCIENCE
YEAR: II.
GROUP: 30422

Table of Contents

Assignment Task

Design and implement a polynomial calculator with a dedicated graphical interface through which the user can insert polynomials, select the mathematical operation (i.e., addition, subtraction, multiplication, division, derivative, integration) to be performed and view the result.

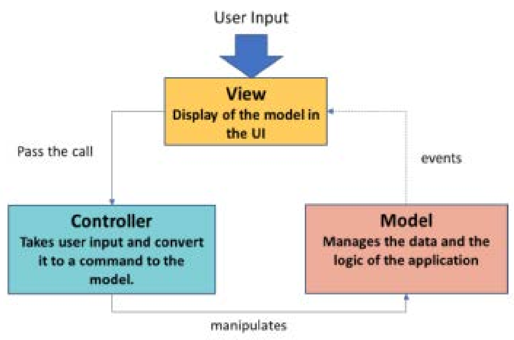
The project should use an object-oriented design(with encapsulation, inheritance, decomposition), should use lists instead of arrays, use foreach instead of for, have a GUI using JavaFX or Swing and every class should have less than 300 lines of code.

An additional requirement is to use an architectural pattern, like MVC. The operations that should be implemented are the following: addition, subtraction, multiplication, division, derivation and integration.

As a bonus, i used RegEx to identify the polynomial, and used JUnit for testing each operation.

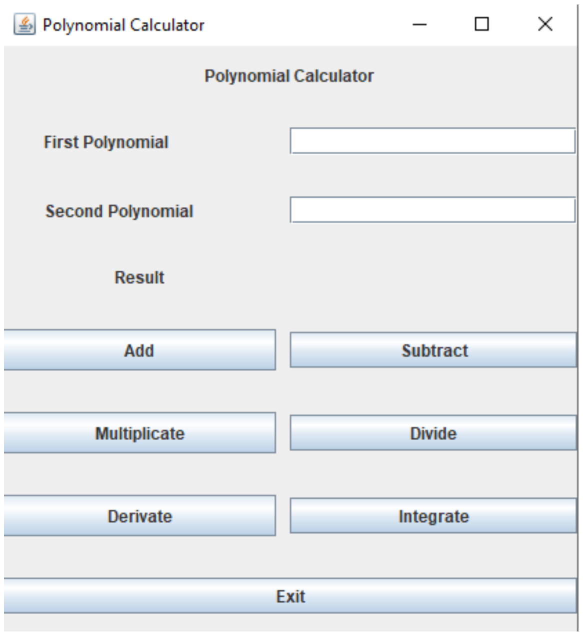
Requirements	**Points**
Object-oriented design ✓	minimal requirement
Lists instead of arrays ✓	minimal requirement
Foreach insted of for ✓	minimal requirement
Java Swing or JavaFX ✓	minimal requirement
Addition and substraction ✓	minimal requirement
Architectural pattern ✓	1 point
Multiplication operation ✓	0.5 point
Derivative operation ✓	0.5 point
Integration operation ✓	0.5 point
RegEx ✓	0.5 point
JUnit ✓	1 point
Division operation ✕	-1 point

Assignment analysis, scenario, approach, and use cases


User Input
View
Display of the model in the UI
Pass the call
events
Controller
Takes user input and convert it to a command to the model.
Model
Manages the data and the logic of the application
manipulates

The first step I took was to create the architectural pattern. I declared the packages neccesary (Model, View and Controller). I also added the Main.java file, which only calls the initialize method from the Controller class.

To introduce you to the MVC architecture design I will talk a bit about it. We have to split all the components of the project into 3 main packages: Model, View and Controller. Each of these elements have a seperate role they have to fulfill. The Model has to manage the data and the logic of the application, the View describes the way the Model is displayed in the UI and the Controller takes user input and converts it into a command that is then forwarded to the Model. This creates a circular motion of logic between the packages: the user interfaces with the View, which then passes the call to the Controller, which manipulates the Model, which in turn creates events to pass back to the View. This approach is really efficient for code re-use and parallel development.



The program itself is quite straightforward to use.

I included two textfields for the user to input the polynomials they want to operate on.

Each monomial included in the polynomial should have their coefficients and degrees described, otherwise a warning window will pop up stating that the input format is wrong.

Example of a correct input : 6x^3+2.3x^2+5x^0

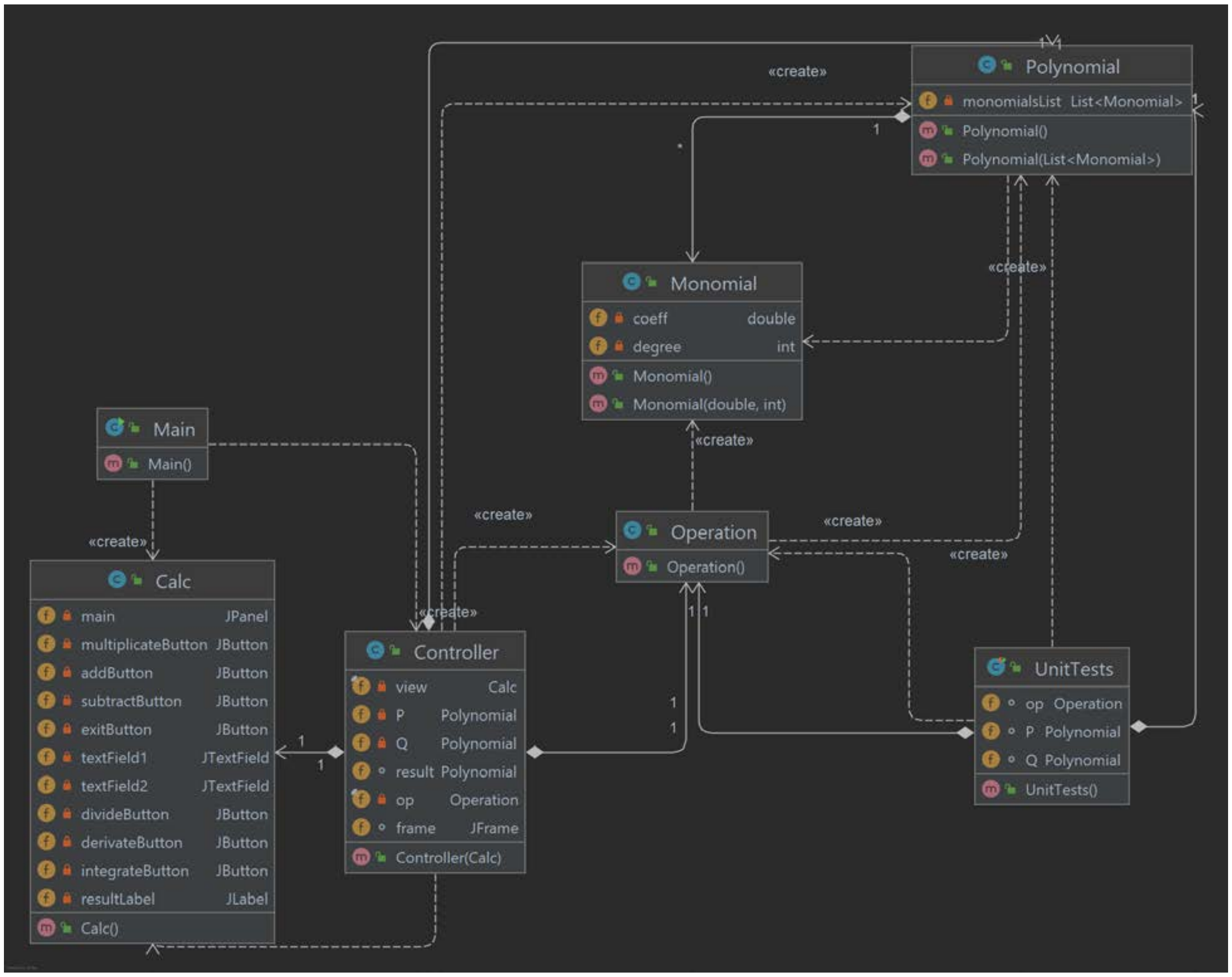
Similarly, if the user wants to do any operation which has to take 2 polynomials, but one of the textfields are empty, they will get a pop-up stating that they didn’t provide the necessary input.

If the user wants to Derivate or Integrate they have to enter the polynomial in the first textfield. Each time those buttons are pressed they will only search in the first textfield.

<u>The required steps to use the program:</u>

1. Start the application
2. Enter the polynomials specifying each coefficient and degree of the monomials
3. Press the desired operation's button
4. Read the answer next to the "Result" label

Projection

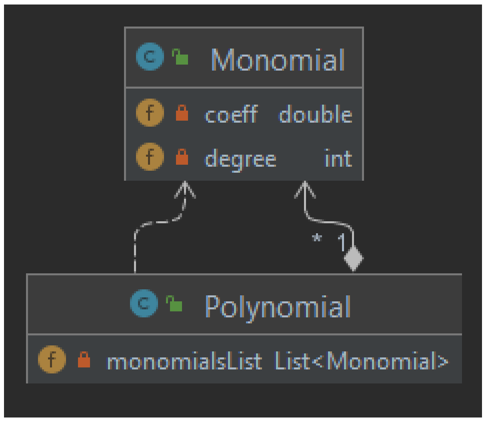


The picture you can see is called a Unified Modelling Language (UML) diagram. It is used to visualize, specify, and document the software system. This is an Implementation level view. As you can see, I implemented quite a few classes to ease the workflow. However complicated it may seem at first glance, it's quite easy to grasp the idea. The <u>Calc</u> class is in the View package, it describes all the components of the User Interface. The Controller package consists of the Operation class and the Controller class. As we can see there is a realisation dependency between the View and Controller packages. The Model package consists of the Monomial and the

Polynomial classes. The polynomial class depends on the Monomial class because a polynomial is a collection of different monomials. I will talk about each of the classes in detail in a bit.

Implementation

The Model



To be sincere, this package was the easiest to design and create. It consists of two classes, Monomial and Polynomial. As I said before, the Polynomial class depends on the other, by definition. To follow the Object-Oriented paradigm, all of the attributes are private. If we want to access them we need getters and setters.

The code below is the implementation of the Monomial class. It has two attributes, coefficient and degree. I have 2 constructors for this class, the first one takes as arguments both the coefficient and the degree, and the second one takes no arguments – but initializes the attributes to 0.

```
public class Monomial {
    private double coeff;
    private int degree;

    public Monomial(double coeff, int degree) {
        this.coeff = coeff;
        this.degree = degree;
    }

    public Monomial() {
        this.degree = 0;
        this.coeff = 0;
    }

    public double getCoeff() {
        return coeff;
    }

    public void setCoeff(double coeff) {
        this.coeff = coeff;
    }

    public int getDegree() {
        return degree;
    }
```

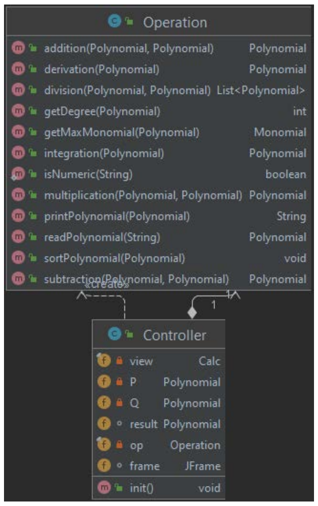
```
  public void setDegree(int degree) {
      this.degree = degree;}}
```

The Polynomial class depends on the Monomial class, as I told you before. It has a single attribute, namely a List of Monomials. Something to note is that I have two different constructors for this class too, the first one takes a list as a parameter and assigns it to the Polynomial, and the second one simply creates an empty list.

```
public class Polynomial {

  private List<Monomial> monomialsList = new ArrayList<>();

  public List<Monomial> getMonomialsList() {
      return monomialsList;
  }

  public void setMonomialsList(List<Monomial> monomialsList) {
      this.monomialsList = monomialsList;
  }


  public Polynomial(List<Monomial> monomialsList) {
      this.monomialsList = monomialsList;
  }

  public Polynomial() {
  }
}
```

The Controller

This is where the user input is converted to the desired output. I have implemented 2 classes in this package too.


Operation
addition(Polynomial, Polynomial) Polynomial
derivation(Polynomial) Polynomial
division(Polynomial, Polynomial) List<Polynomial>
getDegree(Polynomial) int
getMaxMonomial(Polynomial) Monomial
integration(Polynomial) Polynomial
isNumeric(String) boolean
multiplication(Polynomial, Polynomial) Polynomial
printPolynomial(Polynomial) String
readPolynomial(String) Polynomial
sortPolynomial(Polynomial) void
subtraction(Polynomial, Polynomial) Polynomial
«create»
1
Controller
view Calc
P Polynomial
Q Polynomial
result Polynomial
op Operation
frame JFrame
init() void

The Operation package includes every single method that is used in the project. It includes the logic to read, print, add, subtract, etc. the polynomials.

An interesting functionality of this class is the way the program converts a text to a polynomial. I use 2 different methods to achieve this. The first method is just a simple RegEx decision, it checks whether a String is numeric, or not.

```
public static boolean isNumeric(String str) {
    return str.matches("[+-]*\\d*\\.?\\d+");
}
```

The other method is where the fun’s at. It uses RegEx pattern matching too, but includes logic to group the input.

```java
public Polynomial readPolynomial(String text) {
   Polynomial returnPolynomial = new Polynomial();
   String monomialFormat = "([+-]?[\\d\\.]*[a-zA-Z]?\\^?\\d*)";
   String monomialPartsFormat = "([+-]?[\\d\\.]*)([a-zA-Z]?)\\^?(\\d*)";

   Pattern p1 = Pattern.compile(monomialFormat);
   Matcher m1 = p1.matcher(text);

   while (!m1.hitEnd()) {
      m1.find();
      Pattern p2 = Pattern.compile(monomialPartsFormat);
      Matcher m2 = p2.matcher(m1.group());

      if (m2.find()) {

         double coefficient;
         try {
            String coef = m2.group(1);
            if (isNumeric(coef)) {
               coefficient = Float.valueOf(coef);
            } else {
               coefficient = Float.valueOf(coef + "1");
            }
         } catch (IllegalStateException e) {
            coefficient = 0.0F;
         }

         int degree;
         try {
            String exp = m2.group(3);
            if (isNumeric(exp)) {
               degree = Integer.valueOf(exp);
            } else {
               degree = 1;
            }
         } catch (IllegalStateException e) {
            degree = 0;
         }
         if (coefficient == 0.0F)
            return null;
         else {
            Monomial insert = new Monomial(coefficient, degree);
            returnPolynomial.getMonomialsList().add(insert);
         }
      }
   }
   return returnPolynomial;
}
```

I wan to show how an operation is performed. I will use the addition method for example, but every other method implementing an operation works in a similar way.

This method gets the two polynomials as input and returns the result as an output. I have two foreach cycles running through the MonomialsList of both polynomials. I check on each loop if the degrees of the monomials match, and if they do, I will add the coefficient of the first polynomial to the coefficient of the second one. If there's no match, I simply add the monomial to the result's monomial list.

```
public Polynomial addition(Polynomial P, Polynomial Q) {
    Polynomial returnPolynomial = new Polynomial();
    returnPolynomial.setMonomialsList(P.getMonomialsList());
    for (Monomial i : Q.getMonomialsList()) {
        boolean found = false;
        for (Monomial j : returnPolynomial.getMonomialsList()) {
            if (i.getDegree() == j.getDegree()) {
                j.setCoeff(j.getCoeff()+i.getCoeff());
                found = true;
            }
        }
        if (!found)
            returnPolynomial.getMonomialsList().add(i);
    }
    return returnPolynomial;
}
```

The Controller class includes more subclasses, which are necessary for the buttons to function. It has an initialization method, which describes the properties of the main frame. This method describes the content that must be shown, the required behaviour of the exit button and the size of the window.

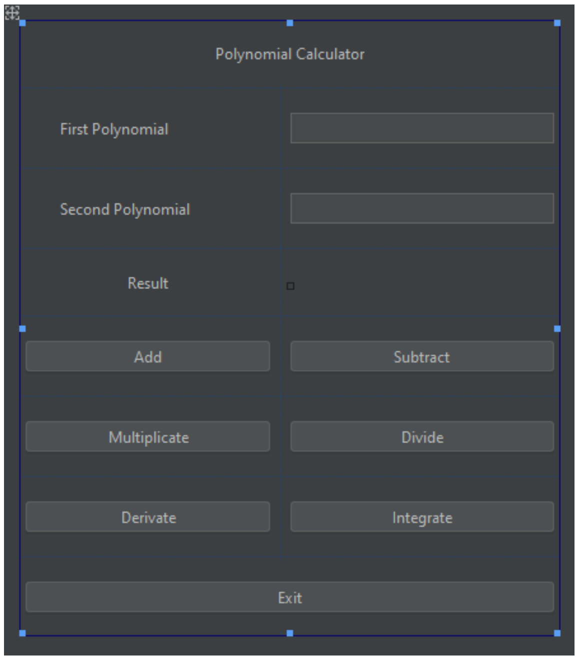
```
public void init() {
    frame.setContentPane(view.getMain());
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(432, 463);
    frame.setVisible(true);
}
```

As I said before, each button has its own subclass. As an example, let me present the Addition Button.

All the buttons use the same logic. First, the program checks if the user provided any input. If they did not, a warning message will be shown. If any input was provided the program tries to interpret them. Any errors or irregularities in the input format will result in a null polynomial, and in turn in a warning message. If everything is correct, the result is calulated and output next to the Result label.

```
class AddButton implements ActionListener {
   @Override
   public void actionPerformed(ActionEvent e) {
      if (view.getTextField1().getText().equals("") || view.getTextField2().getText().equals("")) {
         JOptionPane.showMessageDialog(view.getMain(), "Bad input format");
      } else {
         P = op.readPolynomial(view.getTextField1().getText());
         Q = op.readPolynomial(view.getTextField2().getText());
         if (P == null || Q == null) {
            JOptionPane.showMessageDialog(view.getMain(), "Bad input format");
         } else {
            result = op.addition(P, Q);
            view.getResultLabel().setText(op.printPolynomial(result));
         }
      }
   }
}
```

The View



This package has only one class, namely the Calc class. I created the User Interface with IntelliJ’s built-in GUI creation tool. It uses Java Swing and has an intuitive workflow. Considering how a calculator looks like in real life, I decided to use the GridLayoutManager provided with IntelliJ to create a grid where I can place every element.

IntelliJ creates two separate files when using this tool, a .form and a .java file.

The .form file is the one where you can drag the elements, rename them, shape them to your needs. Sometimes the software does not want to cooperate with the developer which can be frustrating, but it is what it is.

The java file includes all the objects which we created in the form, and methods for each button. We need these methods to call and pass the ActionListeners.

```
public class Calc extends JFrame{
    private JPanel main;
    private JButton multiplicateButton;
    private JButton addButton;
    private JButton subtractButton;
    private JButton exitButton;
    private JTextField textField1;
    private JTextField textField2;
    private JButton divideButton;
    private JButton derivateButton;
    private JButton integrateButton;
    private JLabel resultLabel;

    get&set

    buttons

    public Calc() {    }
}
```

I will not show you the getters and setters at this point, because we already know how they work, but I will show you how a button's method looks like.

```
public void AdditionButton(ActionListener actionListener) {
    addButton.addActionListener(actionListener);
}
```

Results

After I implemented every functionality of the program I used JUnit to test its funtionality. For this to happen, the Java project had to be a Maven Project, and I had to modify the pom.xml file. Here, I specified the dependencies required to use JUnit.

```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.8.2</version>
    </dependency>
</dependencies>
```

After the package was added, I created a new Class in the Controller package (becuase there is every operation i want to test) and started writing the test functions. In order to run a JUnit test, two packages have to be imported: org.junit.jupiter.api.Test and org.junit.api.Assertions.assertTrue. The way I tested the methods is the same way a user would interface with the program. I provided text input, which the program has to interpret correctly. Then I call the JUnit test, to check if the operation is executed in a right way. If any of the methods don’t work, the test will return a False value, meaning that my implementation is not correct.

Example: the method to test the addition operation

```java
package Controller;

import Model.Polynomial;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertTrue;

public class UnitTests {
   Operation op = new Operation();
   Polynomial P = new Polynomial();
   Polynomial Q = new Polynomial();

   @Test
   public void addTest()
   {
      P=op.readPolynomial("3x^2+2x");
      Q=op.readPolynomial("x^2-x");
      assertTrue(op.printPolynomial(op.addition(P,Q)).equals("+4.0x^2+1.0x^1"));
   }
}
```

Possible further development

Unfortunately I could not get the division operation to work. I have implemented it, and the algorithm I used is correct as far as I know, but it simply didn’t want to work.

If I can have a humble request and ask any superior to check on my code, I would really appreciate it. All the logic is there, it just doesn’t want to run properly.

Conclusion

This polynomial calculator can take any two polynomials and add them, subtract them, multiply them. If we want to derivate or integrate a polynomial it is also possible. The GUI is easy to understand and the way we have to provide the polynomials is self explanatory. The project uses many object-oriented paradigms and is easy to develop and work on in the future.

This assignment proved to be useful in helping me to grasp the idea of Object-Oriented programming, encapsulation, and the MVC architecture.

Unit testing with JUnit was a new thing to me, and I think this was the perfect way to get introduced to it.

I found translating mathematical notions and operations into code really fun and challenging 😊.

Bibliography

Course slides

Assignment documentation

Support presentation

https://dsrl.eu/courses/pt/

https://www.geeksforgeeks.org/

https://www.youtube.com/

https://stackoverflow.com/