DOCUMENTATIE

TEMA *1*

NUME STUDENT: BOTĂNEL MIRONA
GRUPA: 30224

CUPRINS

1. Obiectivul temei

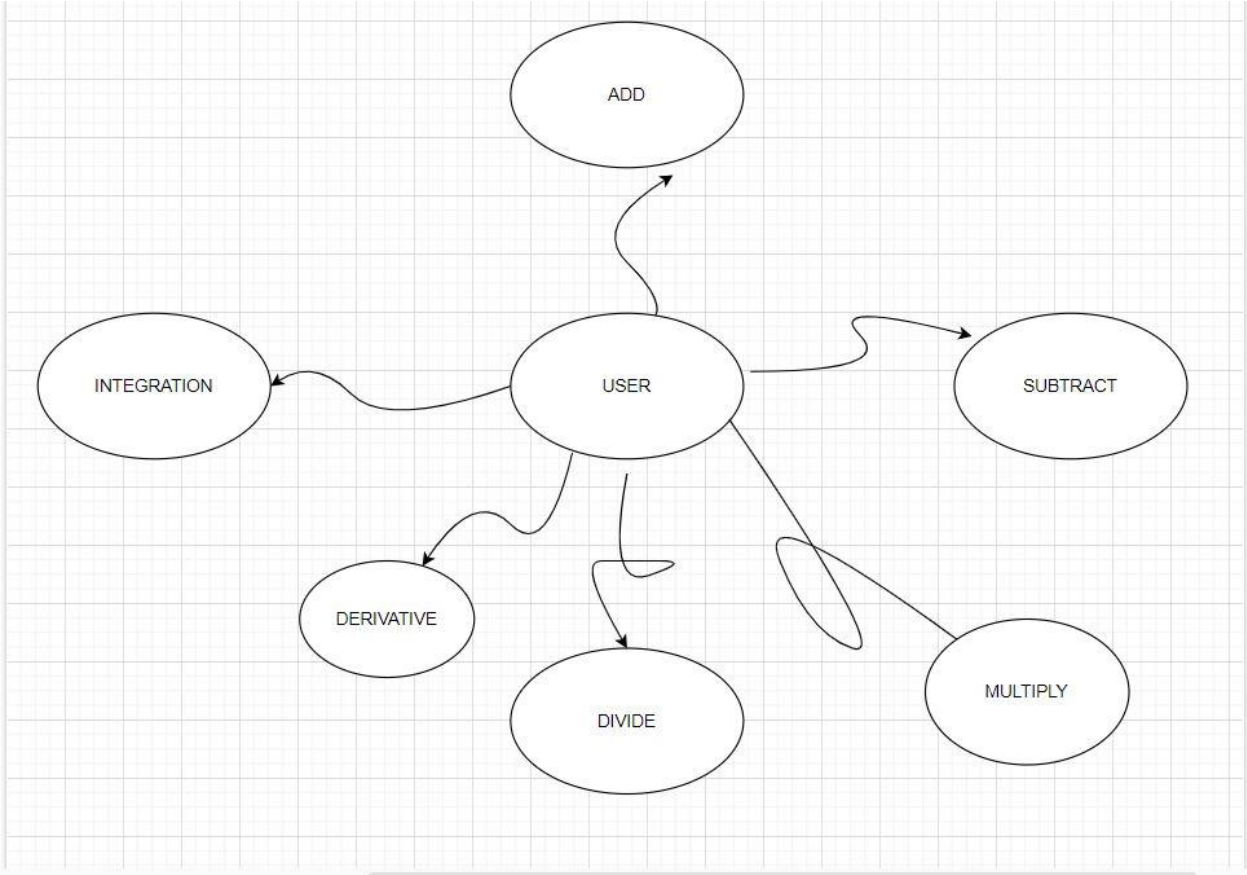
1.1 OBIECTIV PRINCIPAL

Proiectarea și implementarea unui calculator de polinoame cu o interfață grafică prin intermediul careia utilizatorul poate introduce polinoame, selecta operația matematică (adunare, scădere, înmulțire, împărțire, derivare, integrare) care trebuie efectuata și vizualiza rezultatul.

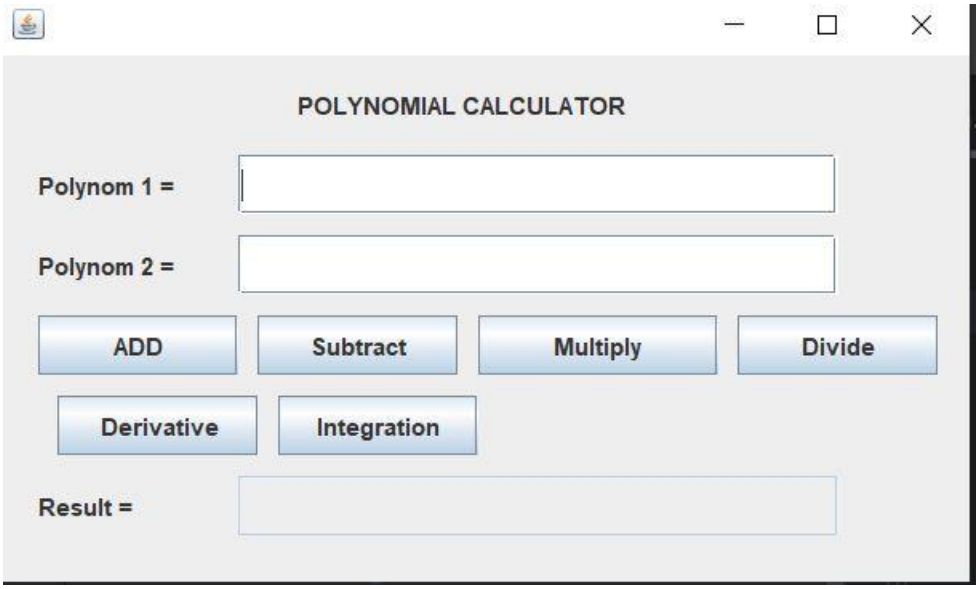
1.2 OBIECTIVE SECUNDARE

Realizarea USE CASE-urilor și a scenariilor.
Împărțirea proiectului pe clase(pentru construirea polinomului, pentru operații, pentru interfață)
Folosirea algoritmilor pentru implementarea și prelucrarea polinoamelor
Testarea

2. Analiza problemei, modelare, scenarii, cazuri de utilizare



-Programul este conceput pentru adăugarea a 2 polinoame pentru operatiile cu 2 intrări, iar, la operatiile ce necesită o singura intrare, input-ul va trebui scris in text box-ul primului polinom. În cazul în care se tastează un input ce nu este polinom, va apărea mesajul: "Invalid polynomial".
-După ce au fost introduse datele, se apasă butonul dorit. Fiecare buton este specific unei operații.
-Interfața calculatorului este una simplă de utilizat, dar totuși clară și explicită:



3. Proiectare

- *STRUCTURI DE DATE*

Pentru realizara proiectului s-a folosit structura de date HashMap. Pentru cheie s-a utilizat exponentul unui monom, iar pentru valoare, coeficientul.

- *DESIGN*

Proiectul conține 3 pachete: GUI (ce contine toate clasele responsabile cu interfața grafica a proiectului), MODEL (ce contine clasele Monomial si Polinom) si LOGIC(ce conține clasa Operations, responsabilă cu toate operațiile ce au loc pe polinoame).

- *DIAGRAMA DE CLASE*

Unified Modeling Language (UML) este un limbaj standard pentru descrierea de modele și specificații pentru software. Este folosit pentru reprezentarea claselor și a relațiilor sintre acestea. Diagramele de clase conțtin numele claselor, variabilele instanță si metodele implementate.

- *ALGORITMII FOLOSIȚI*

Algoritmii folosiți în acest proiect sunt pentru efectuarea operațiilor pe polinoame(adunare, scădere, înmulțire, împărțire, derivare, integrare)

- ✓ *ADUNARE*

```
public  static Polinom addPolynomials (Polinom p1, Polinom p2)
{
    Map<Integer, Monomial> result = new HashMap<>(p1.getPolinom());
    for(Map.Entry<Integer, Monomial> entry : p2.getPolinom().entrySet()) {
        int exponent = entry.getKey();
        double coefficient = entry.getValue().getCoeficient();
        if(result.containsKey(exponent))
        {
            double oldCoefficient = result.get(exponent).getCoeficient();
            if(oldCoefficient + coefficient !=0 ){
            result.put(exponent, new Monomial(exponent,  coeficient: oldCoefficient + coefficient));
            }
        }
        else {
            result.put(exponent, new Monomial(exponent, coefficient));
        }
    }
    return new Polinom(result);
}
```

✓ *SCĂDERE*

```java
public static Polinom subtract(Polinom p1, Polinom p2) {
    Map<Integer, Monomial> result = new HashMap<>();

    // Adăugăm termenii din p1 in rezultat
    for (Map.Entry<Integer, Monomial> entry : p1.getPolinom().entrySet()) {
        int exponent = entry.getKey();
        double coefficient = entry.getValue().getCoeficient();
        result.put(exponent, new Monomial(exponent, coefficient));
    }

    // Scădem termenii din p2 din rezultat
    for (Map.Entry<Integer, Monomial> entry : p2.getPolinom().entrySet()) {
        int exponent = entry.getKey();
        double coefficient = entry.getValue().getCoeficient();

        // Dacă există deja un termen cu aceeasi putere în rezultat, scădem coeficientul
        if (result.containsKey(exponent)) {
            double oldCoefficient = result.get(exponent).getCoeficient();
            double newCoefficient = oldCoefficient - coefficient;
            result.put(exponent, new Monomial(exponent, newCoefficient));
        } else {
            // Dacă nu există un termen cu aceeasi putere în rezultat, adăugăm termenul negativ
            result.put(exponent, new Monomial(exponent, -coefficient));
        }
    }
    // Eliminăm termenii cu coeficientul zero din rezultat
    result.entrySet().removeIf(entry -> entry.getValue().getCoeficient() == 0);

    return new Polinom(result);
}
```

✓ *ÎNMULȚIRE*

```
public static Polinom multiply (Polinom p1, Polinom p2) {
    Map<Integer, Monomial> result = new HashMap<>();

    for (Map.Entry<Integer, Monomial> entry1 : p1.getPolinom().entrySet()) {
        int exponent1 = entry1.getKey();
        double coefficient1 = entry1.getValue().getCoeficient();

        for (Map.Entry<Integer, Monomial> entry2 : p2.getPolinom().entrySet()) {
            int exponent2 = entry2.getKey();
            double coefficient2 = entry2.getValue().getCoeficient();

            int newExp = exponent1 + exponent2;
            double newCoefficient = coefficient1 * coefficient2;

            if (result.containsKey(newExp)) {
                double oldCoef = result.get(newExp).getCoeficient();
                result.put(newExp, new Monomial(newExp, coeficient: oldCoef + newCoefficient));
            } else {
                result.put(newExp, new Monomial(newExp, newCoefficient));
            }
        }
    }

    return new Polinom(result);
}
```

✓ *ÎMPĂRȚIRE*

```java
public static String divide(Polinom dividend, Polinom divisor) {
    if (divisor.isZero()) {
        System.out.println("EROARE. NU AI VOIE SA IMPARTI LA 0");
        return null;
    }

    Polinom result = new Polinom();
    Polinom remainder = new Polinom(dividend.getPolinom());

    while (true) {
        Monomial highestDividendTerm = remainder.getHighestTerm();
        Monomial highestDivisorTerm = divisor.getHighestTerm();

        if (highestDividendTerm == null || highestDividendTerm.getExponent() < highestDivisorTerm.getExponent()) {
            break;
        }

        double newCoefficient = highestDividendTerm.getCoeficient() / highestDivisorTerm.getCoeficient();
        int newExponent = highestDividendTerm.getExponent() - highestDivisorTerm.getExponent();

        Map<Integer, Monomial> term = new HashMap<>();
        term.put(newExponent, new Monomial(newExponent, newCoefficient));
        Polinom termPolynomial = new Polinom(term);
        Polinom termResult = Operations.multiply(termPolynomial, divisor);

        result = Operations.addPolynomials(result, termPolynomial);
        remainder = Operations.subtract(remainder, termResult);
    }

    String sb = "Q: " + result.toString() + "; R: "+ remainder.toString();
    return sb;
}
```

✓ *DERIVARE*

```java
public static Polinom derivative (Polinom polinom)
{
    Map<Integer, Monomial> result = new HashMap<>();
    for(Map.Entry<Integer, Monomial> entry : polinom.getPolinom().entrySet()) {
        int exponent = entry.getKey();
        double coefficient = entry.getValue().getCoeficient();
        coefficient = coefficient * exponent;
        if(exponent != 0 )
        {
            exponent -- ;
        }
        result.put(exponent, new Monomial(exponent, coefficient));
    }
    return new Polinom(result);
}
```

✓ *INTEGRARE*

```
public static Polinom integration (Polinom polinom)
{
    Map<Integer, Monomial> result = new HashMap<>();
    for(Map.Entry<Integer, Monomial> entry : polinom.getPolinom().entrySet()) {
        int exponent = entry.getKey();
        double coefficient = entry.getValue().getCoeficient();
        if(exponent!=-1)
        {
            double newCoefficient = coefficient / (exponent+1);
            result.put(exponent+1, new Monomial( exponent: exponent+1, newCoefficient));
        }
        else {
            result.put(-1, new Monomial( exponent: -1, coefficient));
        }
    }
    return new Polinom(result);
}
```

4. Implementare

- *CLASA MONOMIAL*

*Un monom este un termen al polinomului (ex: a*x^2).*
Clasa Monom contine un exponent de tip int și coeficientul, de tip double, plus metodeleȘ getExponent, getCoeficient, setExponent și setCoeficient.

- *CLASA POLINOM*

În această clasă se realizeaza PARSAREA, adică din String-ul din textBox-ul de la input, va rezulta un polinom. Un polinom, practic, este o lista de monoame. Cea mai importantă metodă este însuși constructorul (public Polinom(String input)), in cadrul acestuia realizându-se parsarea.

```
public Polinom(String input)  {
    Map<Integer, Monomial> result = new HashMap<>();
    Pattern pattern = Pattern.compile("((-?\\d+(?=x))?(-?[x])(\\^(-
?\\d+))?)|((-?)[x])|(-?\\d+)"); //string ul respecta regula:  "ax^b", "+/-
x^b", "+/-x", sau constante simple.
    input = input.replaceAll("\\s", "");
    input = input.replaceAll("\\*", "");
    Matcher matcher = pattern.matcher(input);
    double coefficient = 0;
    int exponent = 0;
    boolean isValid = false;
    while (matcher.find()) {
        isValid = true;
        if (matcher.group(3) != null && matcher.group(2) != null) { //the
monomial of the form ax^b

            exponent = (matcher.group(5) != null ?
Integer.parseInt(matcher.group(5)) : 1);
            coefficient = Integer.parseInt(matcher.group(2));
        } else if (matcher.group(3) != null && matcher.group(2) == null) {
//the monomial of the form +/-x^b or +/-x
            if (matcher.group(3).equals("-x")) {
                coefficient = -1;
```

```
            } else {
                coefficient = 1;
            }
            exponent = (matcher.group(5) != null ?
Integer.parseInt(matcher.group(5)) : 1);
        } else if (matcher.group(3) == null && matcher.group(2) == null) {
//the constant monomial, without x
            coefficient = Integer.parseInt(matcher.group());
        }
        result.put(exponent, new Monomial(exponent, coefficient));
        coefficient = 0;
        exponent = 0;
    }
    if(!isValid) {
        JOptionPane.showMessageDialog(null, "Invalid polynomial\n", "Error",
JOptionPane.ERROR_MESSAGE);
        this.polinom.clear();
    }
    this.polinom = result;
}
public boolean isZero () {
    //verificam daca polinomul e un nr ct
    return polinom.size() == 1 && polinom.containsKey(0) &&
polinom.get(0).getCoeficient()==0;
}
public int degree () {
    if (isZero()){
        return 0;
    }
    int maxExponent = Integer.MAX_VALUE;
    for(int exponent : polinom.keySet()){
        if(exponent > maxExponent)
        {
            maxExponent = exponent;
        }
    }
    return maxExponent;
}
```

- *CLASA OPERATIONS*

În această clasă se realizeaza operațiile pe polinoame, algoritmii utilizați fiind cei prezentați și anterior.

Metoda de adunare (Polinom p1, Polinom p2) primește ca date de intrare 2 polinoame. Se va contrui un nou HashMap, în care, prima dată, vom pune polinomul 1. Pentru fiecare monom din polinomul 2 se verifica daca exponentul lui are un corespondent în result (polinomul rezultat). Dacă da, se adaugă la coeficientul vechi corespunzător exponentului, coeficientul cel nou. Daca nu, se adaugă un nou monom la result.

Metoda de scădere (polinom 1, polinom 2) este implementată asemănător: se adaugă p1 la result, după care, scădem pe p2, analog modului în care am adăugat termenii la adunare.

Metoda de înmultire a 2 polinoame funcționează pe același principiu, dar se va înmulți fiecare termen din primul polinom, cu fiecare termen din cel de-al 2-lea polinom (=: exponenții se vor aduna, coeficientii se vor înmulși)

Metoda de împărțire a 2 polinomi este cea mai deosebitaŞ Ea primeste, ca date de intrare, 2 polinoame (dividend, divisor). Daca divisor = 0 =: eroare, deoarece nu se poate împărți la 0. Daca nu, se va tine minte, la fiecare pas, câtul (result) și restul (remainder). Cât timp gradul dividentului este mai mare decât cel al lui divisor, sau este diferit de 0, se va calcula noul coeficient, ca fiind coeficientul termenului cu cel mai mare exponent de la divident / coeficientul termenului cu cel mai mare exponent de la divisor. Analog se va face si pt noul exponent, dar nu se va împărți, ci se va scîdea. Polinomul result va fi egal cu suma dintre vechiul result și produsul dintre monomul cu exponentul și coeficientul nou formați și divisor. Polinomul remainder va fi diferețșa dintre vechiul

remainder și produsul calculat anterior. La final, vom face un String ce va conțineȘ Q (de la cât) + result + R (de la rest) + remaind.Se va returna String ul obținut.

- *CLASA DESIGN*

  Interfața pentru utilizator.Aici sunt prezentate 6 butoane si 3 Jtextfiled-rui(pentru introducerea primului polinom, celui de-al doilea polinom, iar, ăn al 3-lea, se va afișsa rezultatul).

- *CLASA CONTROLLER*

  Aici se leaga operațiile din Operations de clasa Design, cu ajutorul claselor:

```
public class AddButtonListener implements ActionListener
public class DerivativeButtonListener implements ActionListener
public class DivideButtonListener implements ActionListener
public class IntegrationButtonListener implements ActionListener
public class MultiplyButtonListener implements ActionListener
public class SubtractButtonListener implements ActionListener
```

Mai jos voi lăsa un exemplu:

```
package org.example.gui;

import org.example.gui.Design;
import org.example.logic.Operations;
import org.example.model.Polinom;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Controller {
    private Design design;

    public Controller(Design design) {
        this.design = design;
        // Adăugați ascultători pentru butoanele din interfață
        design.getAddBtn().addActionListener(new AddButtonListener(design));
        design.getSubtractBtn().addActionListener(new
SubtractButtonListener(design));
        design.getMultiplyBtn().addActionListener(new
MultiplyButtonListener(design));
         design.getDivideBtn().addActionListener(new
DivideButtonListener(design));
         design.getDerivativeBtn().addActionListener(new
DerivativeButtonListener(design));
         design.getIntegrationBtn().addActionListener(new
IntegrationButtonListener(design));
    }



    // Definiți ascultătorii pentru celelalte operații așa cum este necesar
    // private class MultiplyButtonListener implements ActionListener { ... }
    // private class DivideButtonListener implements ActionListener { ... }
    // private class DerivativeButtonListener implements ActionListener { ...
}
    // private class IntegrationButtonListener implements ActionListener {
... }
}
```

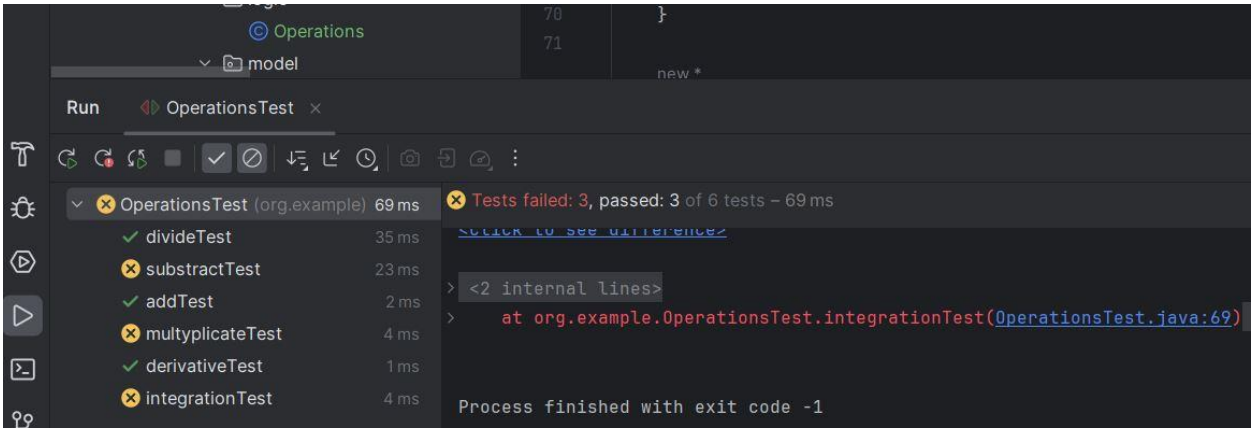
```java
package org.example.gui;

import org.example.logic.Operations;
import org.example.model.Polinom;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class AddButtonListener implements ActionListener
{
    private Design design;
    public AddButtonListener(Design design) {
        this.design = design;
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        String input1 = design.getPoly1Text();
        String input2 = design.getPoly2Text();
        Polinom p1 = new Polinom(input1);
        Polinom p2 = new Polinom(input2);
        Polinom result = Operations.addPolynomials(p1, p2);
        design.setResultText(result.toString());
    }
}
```

5. Rezultate



Am realizat câte un test pentru fiecare operație contruită, pentru 3 din ele am adăugat raspunsurile corecte, iar, pentru 3, răspunsuri greșite.

6. Concluzii

În urma implementării acestui proiect, mi-am aprofundat cunoștințele în cee ace priveste Structurile de date și structurarea codului pe mai multe clase, respective metode, pentru a fi un cod cât se poate de usor de înțeles. Ca dezvoltări ulterioare, proiectul poate avea implementate și alte operații, de exemplu, aflarea rădacinilor unui polinom, modulul etc.

7. Bibliografie

1. *https://docs.oracle.com/en/java/*
2. *What are Java classes? - www.tutorialspoint.com*