

DOCUMENTATIE

TEMA *1*

NUME STUDENT: HOBAN CRISTIAN MIHAI
GRUPA: 30227

CUPRINS

CUPRINS	2
1. Obiectivul temei	3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare	3
3. Proiectare	4
4. Implementare	6
5. Rezultate	11
6. Concluzii	12
7. Bibliografie	12

1. Obiectivul temei

Obiectivul principal al temei îl reprezintă realizarea unui calculator de polinoame în Java care să fie capabil să realizeze următoarele operații: adunare, scădere, înmulțire, împărțire, derivare și integrare.

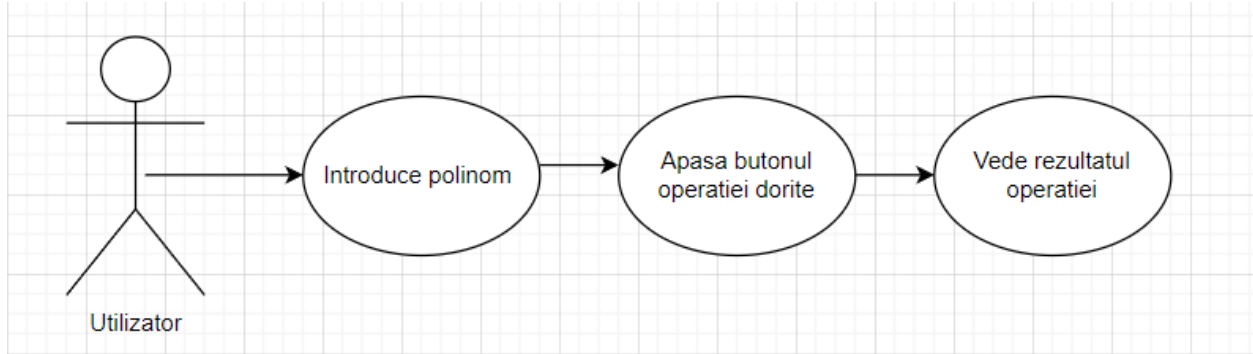
Există mai multe obiective secundare care, realizate, vor duce la atingerea scopului final. Pe acestea le-am enumerat în tabelul de mai jos:

<i>Obiectiv</i>	<i>Descriere</i>	<i>Capitol unde poate fi găsit</i>
1. Definirea unei clasei Polinom	Pentru a permite aplicației să efectueze diferite operații pe polinoame, aceasta trebuie să poată defini un polinom corect	<ul style="list-style-type: none">- Proiectare- Implementare
2. Implementarea operațiilor de adunare, scădere, înmulțire, împărțire și integrare	Operațiile trebuie implementate pentru a oferi funcționalitate aplicației	<ul style="list-style-type: none">- Implementare
3. Crearea unei interfețe grafice	Interfața grafică este necesară pentru ca aplicația să fie cât mai ușor și mai intuitiv de folosit	<ul style="list-style-type: none">- Implementare
4. Validarea datelor de intrare	Pentru a preveni anumite erori, este important ca datele de intrare să fie corecte	<ul style="list-style-type: none">- Implementare
5. Testarea aplicației	Testarea aplicației este necesară pentru a ne asigura că întreaga aplicație funcționează corect	<ul style="list-style-type: none">- Rezultate

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

Aplicația trebuie să permită utilizatorului introducerea de la tastatură a unui sau două (depinde de operație) polinoame, și totodată să permită apăsarea butoanelor specifice operației pe care acesta dorește să o efectueze. Totodată, pentru ușurința introducerii acestor polinoame, o formă generală a acestuia este specificată sub câmpurile de text.

Diagrama use-case:



Pasii pe care utilizatorul trebuie sa ii urmeze:

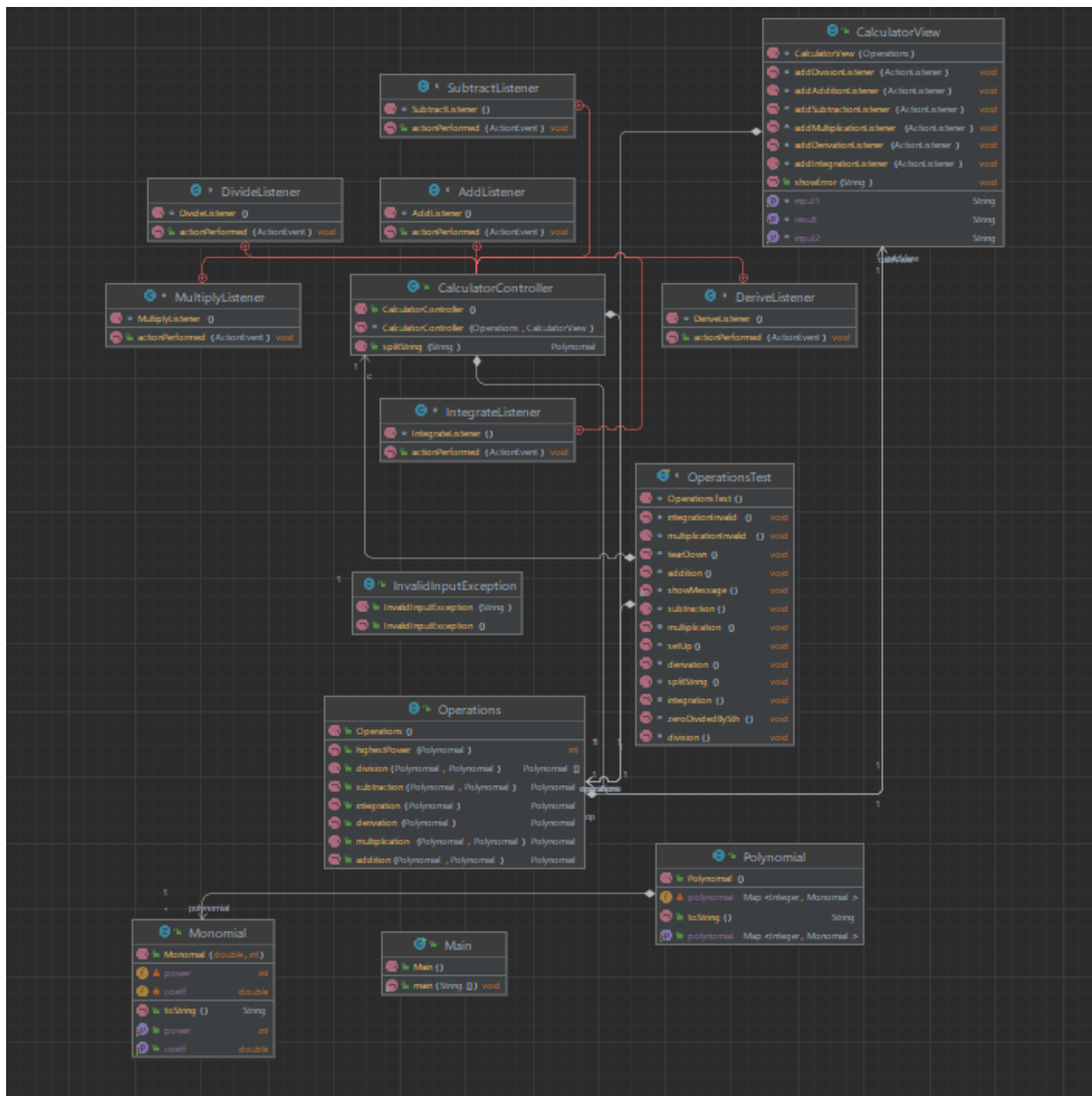
1. Introduce in campurile denumite “Polynomial 1” si “Polynomial 2” polinoamele pe care urmeaza sa se efectueze operatia, respectand forma specificata pe interfata;
2. Apasa butonul cu eticheta specifica operatiei care urmeaza sa fie efectuata(in cazul derivarii si al integrarii, operatiile se fac pe polinomul introdus in campul “Polynomial 1”);
3. Vede rezultatul operatiei efectuate in campul denumit “Result”.

3. Proiectare

Proiectarea OOP reprezinta un model de programare care ofera o buna capacitate de gestionare a unui program.

In acest capitol voi prezenta proiectarea OOP a aplicatiei, in prima faza cu ajutorul unei diagrame UML pentru a fi mai usoara intelegerea modului in care clasele sunt conectate intre ele.

Diagrama UML:



Dupa cum se observa din diagrama, aplicatia este structurata in mai multe clase, fiecare din ele apartinand unui pachet:

- **view**
 - CalculatorView
- **model**
 - Monomial
 - Polynomial
 - Operations
- **controller**
 - CalculatorController (aceasta clasa contine alte 6 clase interne)

- MultiplyListener
 - AddListener
 - DeriveListener
 - SubtractListener
 - IntegrateListener
 - DivideListener
- InvalidInputException
- **org.example**
 - Main

Pentru a implementa polinoamele, am creat intai clasa “Monomial”, in care puteam memora un singur monom, iar apoi am creat clasa “Polynomial” in care am folosit un TreeMap pentru a putea stoca mai multe monoame.

```
public class Polynomial {
    2 usages
    private Map<Integer, Monomial> polynomial;

    no usages
    public Polynomial() { this.setPolynomial(new TreeMap<Integer, Monomial> (Collections.reverseOrder())); }
```

4. Implementare

Dupa cum am specificat si in capitolul anterior, pentru a crea un polinom, am realizat o clasa “Polynomial” in care am folosit un TreeMap de monoame, pe care l-am format in ordine inversa pentru a avea mereu monomanele retinute in ordine inversa puterii lor.

In clasa “**Operations**”, am implementat pe rand pe care calculatorul va trebui sa le poata realiza:

- **addition**
Este o metoda care primeste ca parametri 2 polinoame si returneaza suma acestora. Monoamelor cu puteri comune li se vor aduna coeficientii si se va adauga monomul nou in rezultat, iar monoamele care nu au corespondent in celalalt polinom se vor adauga direct in rezultat;
- **subtraction**
Este o metoda care primeste 2 polinoame ca parametri si returneaza diferenta acestora. Monoanelor cu puteri egale li se vor scadea coeficientii si se vor adauga monoamele formate in rezultat, monomoanele care nu au corespondent din primul polinom care este

considerat descazutul se vor adauga asa cum sunt in rezultat, iar cele din scazator se vor adauga cu semn schimbat;

- **division**

Este o metoda care imparte 2 polinoame dupa algoritmul prezentat in exemplul de mai jos:

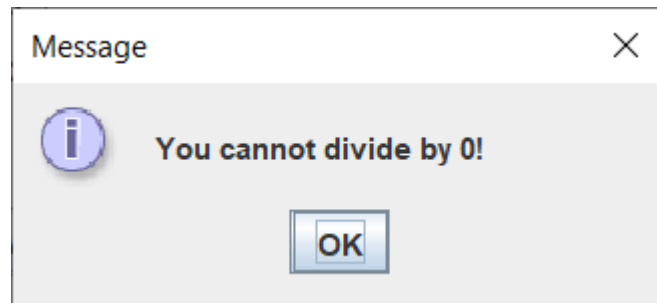
$$\begin{array}{r} (X^3 - 2X^2 + 6X - 5) : (X^2 - 1) = X - 2 \\ \underline{-X^3 \qquad + \quad X} \\ -2X^2 + 7X - 5 \\ \underline{2X^2 \qquad - 2} \\ 7X - 7 \end{array}$$

Quotient = $X - 2$; Remainder = $7X - 7$

Functia primeste ca parametri 2 polinoame si returneaza un vector de 2 polinoame, primul reprezentand catul, iar al doilea restul.

Pentru aceasta metoda am folosit o metoda aditionala, denumita “**highestPower**” pentru a putea determina mereu cea mai mare putere din polinom.

Pentru a elimina orice eroare, am facut o verificare cu privire la impartirea cu 0. Astfel in cazul in care se va incerca acest lucru, aplicatia va afisa un mesaj:



- **multiplication**

Este o metoda care primeste ca parametri 2 polinoame si returneaza produsul acestora. Se inmultesc pe rand monoamele, fiecare cu fiecare (se inmultesc coeficientii si se aduna puterile), iar monomul rezultat, in cazul in care in polinomul care va fi returnat nu exista monom cu puterea acestuia se va adauga, iar in cazul in care exista se va aduna la monomul respectiv din polinomul rezultat noul monom.

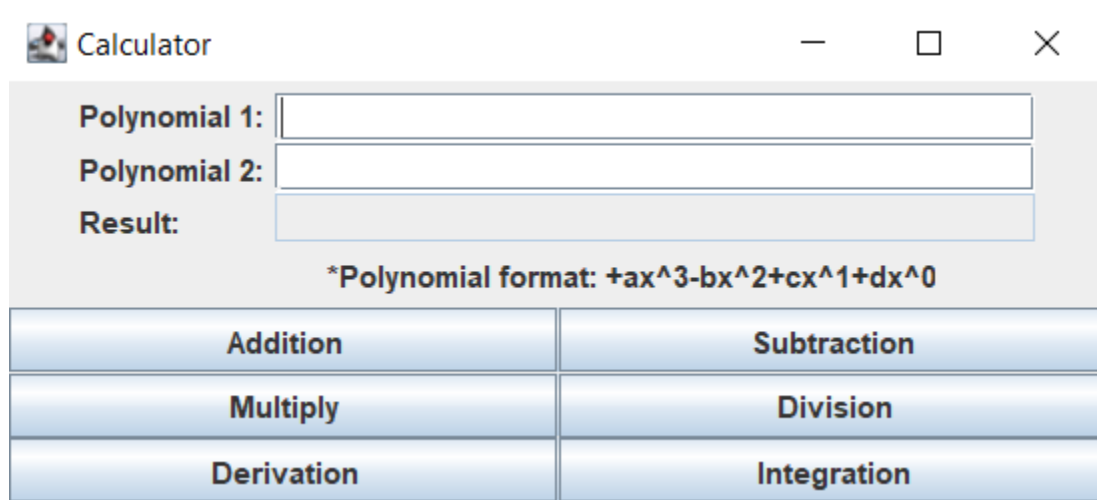
- **integration**

Este o metoda care primeste ca parametru un singur polinom si returneaza valoarea acestuia integrata. Pentru aceasta, luam monom cu monom si impartim coeficientul cu valoarea puterii incrementata cu 1, iar puterea va fi incrementata cu 1.

- **derivation**

Este o metoda care primeste ca parametru un singur polinom si returneaza valoarea acestuia derivata. Pentru aceasta, luam monom cu monom si inmultim coeficientul cu valoarea puterii, iar puterea va fi decrementata cu 1.

In clasa “**CalculatorView**” am implementat interfata grafica a aplicatiei:



Calculator

Polynomial 1:

Polynomial 2:

Result:

*Polynomial format: +ax³-bx²+cx¹+dx⁰

Addition	Subtraction
Multiply	Division
Derivation	Integration

Pentru realizarea acesteia am folosit JTextField-uri pentru campurile de introducere a polinoamelor si pentru campul de afisare a rezultatului, JLabel-uri pentru etichetele acestor campuri si JButton-uri pentru butoanele corespunzatoare fiecarei operatii. Mai exista un JLabel pentru a afisa format-ul unui polinom.

Asezarea in Panel am facut-o astfel:

- Am creat un Panel in care am pus una sub cealalta etichetele “Polynomial 1:”, “Polynomial 2:” si “Result:”, si un alt Panel in care am pus unul sub celalalt cele 3 campuri aferente fiecarei etichete. Aceste 2 Panel-uri le-am pus unul dupa celalalt astfel creand partea de sus a interfetei grafice;
- Am creat un JLabel pentru format-ul polinomului;
- Am creat un Grid cu 3 randuri si 2 coloane in care am introdus cele 6 butoane corespunzatoare fiecarei operatii;
- La final, toate cele 3 parti de mai sus le-am pus in Panel-ul final una sub cealalta.

Pentru a putea face legatura intre interfata grafica si partea functionala a aplicatiei am implementat clasa “**CalculatorController**” in care am creat cate un ascultator pentru fiecare din cele 6 butoane.

Codul de mai jos reprezinta ascultatorul creat pentru butonul de adunare:


```

class AddListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        String userInput1 = "";
        String userInput2 = "";
        Polynomial p = new Polynomial();
        Polynomial p1 = new Polynomial();
        Polynomial p2 = new Polynomial();
        userInput1 = calcView.getInput1();
        userInput2 = calcView.getInput2();
        p1 = splitString(userInput1);
        p2 = splitString(userInput2);
        if (p1 != null && p2 != null) {
            p = operations.addition(p1, p2);
            calcView.setResult(p.toString());
        }
    }
}

```

Metoda “**splitString**”, implementata tot in CalculatorController are rolul de a parsa string-ul introdus de utilizator in monoame care mai apoi vor fi introduse intr-un polinom.

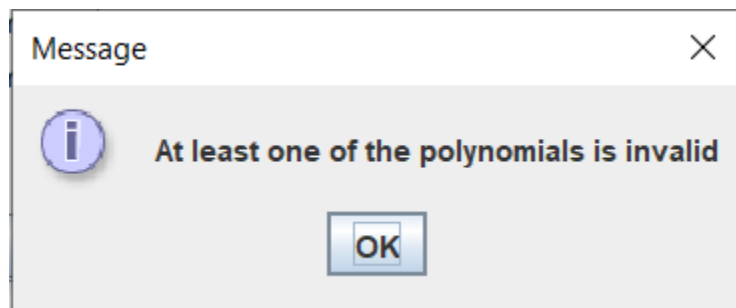
```

Pattern pattern = Pattern.compile( regex: "[+-]\\d*x\\^\\d+");
Matcher matcher = pattern.matcher(s);
String verify = "";
while (matcher.find()) {
    String m = matcher.group();
    String[] v = m.split( regex: "x\\^");
    double coeff = Double.parseDouble(v[0]);
    int power = Integer.parseInt(v[1]);
    Monomial mon = new Monomial(coeff, power);
    if (mon.getCoeff() < 0)
        verify += mon.toString();
    else if (mon.getCoeff() >= 0)
        verify += "+" + mon.toString();

    if(result.getPolynomial().get(mon.getPower()) == null ) {
        result.getPolynomial().put(mon.getPower(), mon);
    }
    else {
        coeff = coeff + (result.getPolynomial().get(mon.getPower()).getCoeff());
        mon.setCoeff(coeff);
        result.getPolynomial().put(mon.getPower(), mon);
    }
}

```

Tot in aceasta metoda am implementat si verificarea corectitudinii polinomului introdus. In cazul in care acesta nu respecta regulile din format-ul afisat pe interfata, aplicatia va afisa un mesaj:



Aceasta verificare functioneaza astfel. Imparte string-ul format in monoame care mai apoi vor fi introduse in polinom. La final, se verifica daca afisarea polinomului format care se face cu metoda **toString()** este echivalenta cu string-ul introdus in campul specific.

```
if (result.toString().equals("0"))
    verify = "0";
try {
    if (!s.equals(verify)) {
        throw new InvalidInputException();
    } else {
        return result;
    }
} catch (InvalidInputException ex) {
    calcView.showError( errorMessage: "At least one of the polynomials is invalid");
}
return null;
```

5. Rezultate

Pentru a fi sigur ca aplicatia functioneaza conform asteptarilor, am facut o testare unitara cu utilitarul JUnit, testand fiecare metoda pe rand.

Spre exemplu, testul pentru adunare arata astfel:

```
@org.junit.jupiter.api.Test
void addition() {
    totalNumber++;
    Polynomial p1 = new Polynomial();
    Polynomial p2 = new Polynomial();
    p1 = c.splitString( s: "+2x^3-4x^1");
    p2 = c.splitString( s: "+6x^3+4x^2");
    String res = "";
    res = op.addition(p1, p2).toString();
    if("+8x^3+4x^2-4x^1".equals(res))
        cnt++;
    assertEquals( expected: "+8x^3+4x^2-4x^1", res);
}
```

6. Concluzii

Pot spune ca pe parcursul realizarii acestui proiect m-am familiarizat cu folosirea structurii de date TreeMap si folosirea functiilor pattern si matcher pentru a parsa un string dupa plac. Am invatat sa folosesc testarea unitara pentru a testa daca aplicatia functioneaza conform asteptarilor.

Ca dezvoltari ulterioare m-am gandit la cateva:

- utilizarea mai complexa a functiilor pattern si matcher pentru a putea introduce si afisa polinoame astfel: in cazul in care un monom are coeficientul 1 sa nu mai afiseze acel 1 in fata sau in cazul in care monomul are puterea 0 sa afiseze doar coeficientul;
- in momentul de fata, coeficientii introdusi de catre utilizator pot fi doar numere intregi. Astfel, s-ar putea implementa astfel incat sa se poata introduce si numere fractionare pentru coeficienti.

7. Bibliografie

1. <https://regexr.com/>
2. <https://dsrl.eu/courses/pt/>