MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
SPECIALIZAREA CALCULATOARE SI TEHNOLOGIA INFORMATIEI

Calculator de Polinoame

-Documentatie-

Ana-Maria Cusco

An academic: 2020 – 2021

Cuprins

Obiectivul Temei

În matematică , un polinom este o expresie formată din variabile (numite și nedeterminate) și coeficienți , care implică doar operațiile de adunare , scădere , multiplicare și exponențiere întreagă non-negativă a variabilelor. Un exemplu de polinom al unui singur x nedeterminat este x^2 - 4 x + 7 . Polinoamele apar în multe domenii ale matematicii și științei. De exemplu, ele sunt folosite pentru a forma ecuații polinomiale , care codifică o gamă largă de probleme, de la probleme de cuvinte elementare la probleme științifice complicate; acestea sunt utilizate pentru a defini funcții polinomiale , care apar în setări variind de la chimie și fizică de bază la economie și științe sociale ; acestea sunt utilizate în calcul și analize numerice pentru a aproxima alte funcții. În matematica avansată, polinoamele sunt folosite pentru a construi inele polinomiale și soiuri algebrice, care sunt concepte centrale în algebră și geometrie algebrică.

Proprietăți elementare ale polinoamelor:

- Suma a două polinoame este un polinom
- Diferenta a două polinoame este un polinom
- Produsul a două polinoame este un polinom
- Impartirea a două polinoame este un polinom
- Derivata unui polinom este un polinom
- Primitiva unui polinom este un polinom [1]

Obiectivul temei il reprezinta dezvoltarea unei aplicatii – un “Calculator de Polinoame”, folosindu-se paradigma programarii orientate pe obiecte , mai specific limbajul Java.

Scopul aplicatiei Java, ce va fi detaliata in cele ce urmeaza, este de a simplifica operatiile cu polinoame, si de a facilita interactiunea cu aceste expresii algebrice utilizand o interfata grafica cat mai prietenoasa pentru utilizator si cat mai simplu de utilizat.

Astfel aplicatia isi propune sa ofere suport pentru realizarea operatiilor cu polinoame cu coeficienti intregi dupa cum urmeaza: adunarea, scaderea, inmultirea, impartirea a doua polinoame precum si integrarea sau derivarea acestora.

Analiza problemei, modelare scenarii, cazuri de utilizare

In analiza problemei am pornit de la cerintele aplicatiei si am incercat sa inteleg domeniul problemei pentru a putea construi domeniul solutiei.

Astfel principalul obiectiv il constituie proiectarea si implementarea unui calculator de polinoame, cu o interfata grafica dedicata care sa-i permita utilizatorului sa insereze polinoame, sa selecteze operatia matematica dorita (adunare, scadere, inmultire, impartire, derivare, integrare) si sa vizualizeze rezultatul.

Pentru a-mi face munca cat mai usoara mai calitativa am decis sa impart problema in mai multe subprobleme, pentru a identifica in prima faza cerintele functionale ale aplicatiei, mai apoi cele nonfunctionale pentru o proiectare cat mai corecta a sistemului, si mai tarziu o implementare usoara, bine documentata.

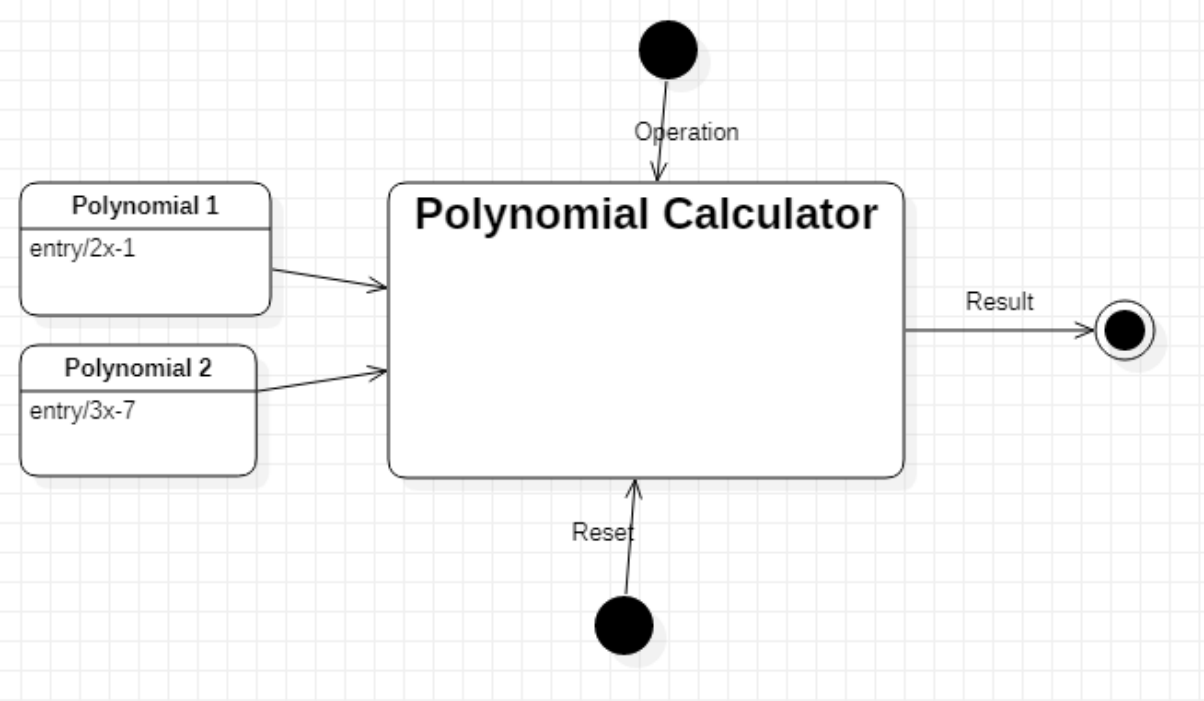
Sub-obiective:

- identificarea cerintelor functionale
- identificarea cerintelor nonfunctionale
- proiectarea solutiei
- implementarea solutiei
- testarea solutiei

Ca si cerinte functionale ale aplicatiei am identificat urmatoarele aspecte: utilizatorul ar trebui sa poata sa introduca in interfata grafica cel putin un polinom pentru operatiile de derivare si integrare si cel putin 2 pentru restul operatiilor, sa poata selecta operatia dorita si sa poata vizualiza rezultatul. Input-urile sunt polinoame cu coeficienti intregi, insa in urma operatiilor de integrare sau impartire se pot obtine polinoame cu coeficienti reali, deci acest detaliu trebuie luat in considerare la proiectare. O alta cerinta functionala o reprezinta partea de tratare a erorilor care pot sa apara in utilizarea incorecta a aplicatiei. Astfel ca utilizatorul poate introduce polinoame care nu sunt valide sau poate incearca sa faca o operatie care necesita 2 termeni si el introduce doar unul. In toate aceste cazuri erorile trebuie semnalate prin mesaje specifice.

Din categoria cerintelor nonfunctionale face parte complexitatea design-ului interfetei grafice, flexibilitatea aplicatiei.

Urmatorul pas pe care l-am facut in dezvoltarea aplicatiei a fost proiectarea unei viziuni in ansamblu a sistemului:

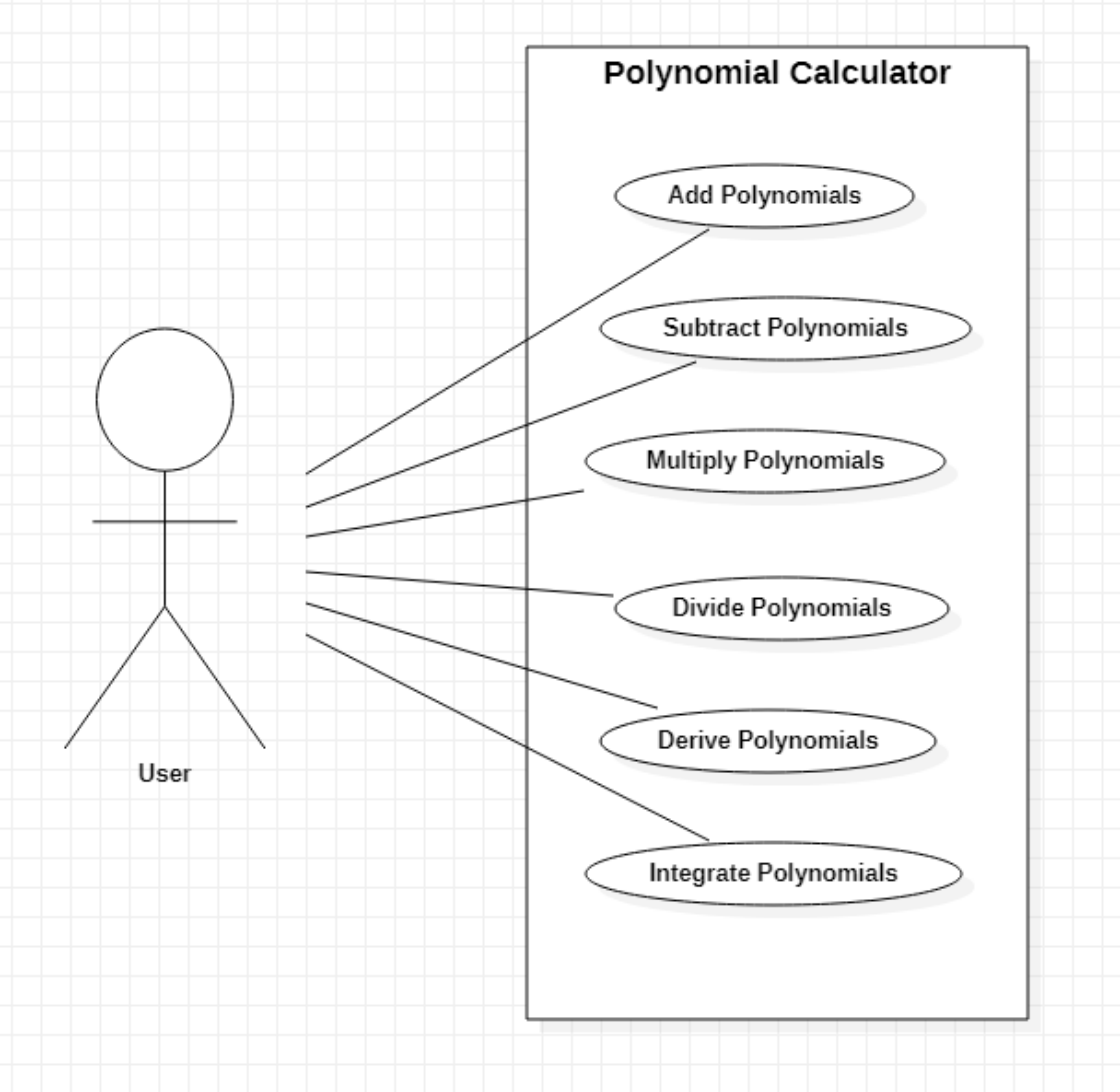


In schema de mai sus sistemul – “Calculatorul de polinoame” este privit ca o “cutie neagra” (eng. black box) care are ca si intrari cele 2 polinoame, operatia matematica dorita si inca o intrare care permite resetarea sistemului (aducerea intr-o stare cunoscuta), in cazul nostru permitandu-i utilizatorului sa efectueze diverse operatii succesiv. Astfel, in urma selectiei operatiei dorite sistemul va fi dus intr-o noua stare unde utilizatorul poate vizualiza rezultatul operatiei sau poate introduce din nou alte intrari.

De asemenea pentru a intelege mai bine functionalitatea si pentru a nu rata detalii importante am definit scenarii si cazuri de utilizare care vor fi detaliate in cele ce urmeaza.

Modelare Scenarii:

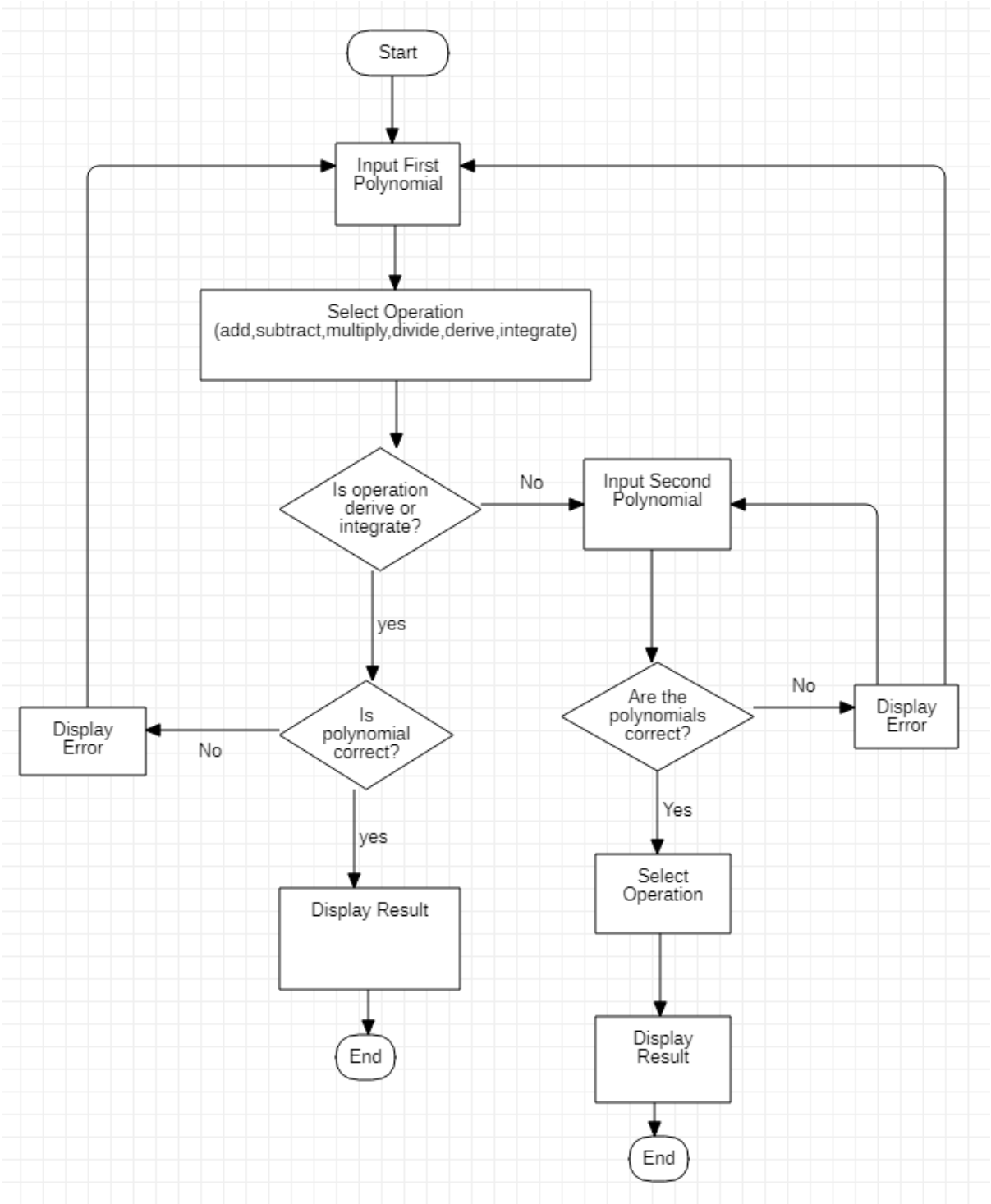
Scenariile de utilizare ale aplicatiei sunt diverse. Trebuie luate in calcul toate cazurile, si mai ales acelea in care “ceva s-ar putea sa mearga prost”. Perspectiva din care se definesc aceste scenarii este cea a utilizatorului obisnuit care interactioneaza cu aplicatia si nu a proiectantului care cunoaste detaliile interne. Utilizatorul ar trebui sa fie avertizat cu mesaje specifice de fiecare data cand utilizeaza incorect aplicatia, iar aplicatia ar trebui sa fie gata sa trateze aceste inconveniente care pot sa apara.



Interactiunea utilizatorului cu aplicatia

Cazuri de utilizare

Cazurile de utilizare le-am incadrat in 2 categorii: cele in care utilizatorul introduce 2 polinoame pentru a efectua o operatie (adunare, scadere, inmultire, impartire) si cele in care se introduce un singur polinom cum sunt derivarea si integrarea. Pentru fiecare dintre cele 2 cazuri se iau unele decizii in funcite de fluxul programului



Flowchart -Polynomial Calculator

Caz de utilizare: Operatii care se efectueaza pe 2 polinoame (adunare, scadere, inmultire, impartire)

Actorul principal: Utilizatorul

Scenariul de succes: Utilizatorul realizeaza operatia dorita.

- ➔ 1. Utilizatorul introduce in interfata grafice 2 polinoame
- ➔ 2. Utilizatorul selecteaza una din urmatoarele operatii (adunare, scadere, inmultire, impartire). Polinomul impartitor in cazul impartirii este diferit de 0!
- ➔ 3. Calculatorul de polinoame realizeaza operatia selectata si afiseaza rezultatul

Scenariul de esec (alternativ celui de success): Polinoamele introduse sunt incorecte!

- ➔ Utilizatorul introduce unul sau doua polinoame incorect.
- ➔ Eroarea corespunzatoare este semnalata (Polinom Incorect. | Impartire cu 0. | Camp necompletat.)
- ➔ Scenariul se intoarce la pasul 1.

Caz de utilizare: Operatii care se efectueaza pe un polinom (derivare, integrare)

Actorul principal: Utilizatorul

Scenariul de succes: Utilizatorul realizeaza operatia dorita.

- ➔ 1. Utilizatorul introduce in interfata grafice cel putin un polinom. In cazul introducerii a doua polinoame calculatorul va afisa rezultatul operatiei pentru primul polinom introdus.
- ➔ 2. Utilizatorul selecteaza una din urmatoarele operatii (derivare, integrare)
- ➔ 3. Calculatorul de polinoame realizeaza operatia selectata si afiseaza rezultatul

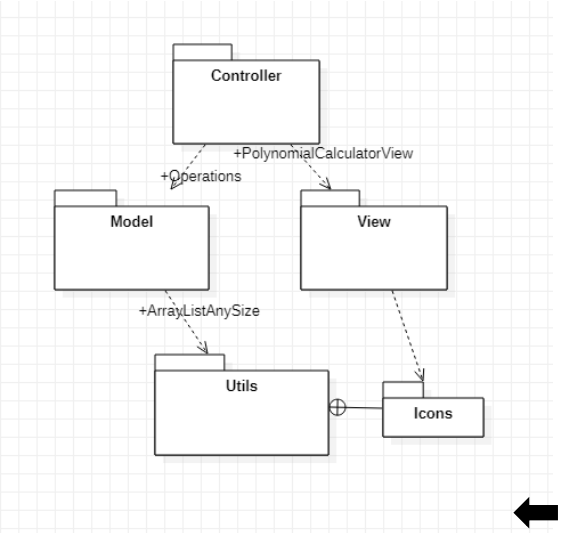
Scenariul de esec (alternativ celui de success): Polinoamele introduse sunt incorecte!

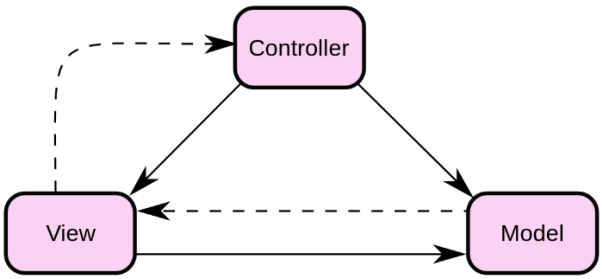
- ➔ Utilizatorul introduce unul sau doua polinoame incorect.
- ➔ Eroarea corespunzatoare este semnalata. (Polinom Incorect!)
- ➔ Scenariul se intoarce la pasul 1.

Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, relatii, packages, algoritmi, interfata utilizator)

Decizii de proiectare

In proiectarea aplicatiei am ales sa folosesc modelul architectural pe mai multe nivele (eng. layers), mai precis arhitectura Model-View-Controller. Acesst tip de abordare mi-a permis sa imi structurez clasele logic in pachete in functie de tip si de utilizare. Pe langa cele trei pachete se mai poate observa si pachetul Utils care contine o clasa ArrayListAnySize pe care am folosit-o in construitrea modelului, aceasta fiind de fapt o clasa descendenta a clasei ArrayList. Tot in acest pachet se gaseste un subpachet care contine iconitele folosite in realizarea interfetei grafice.





Source: https://ro.wikipedia.org/wiki/Model-view-controller

Diagrama UML de pachete

Diagrama UML de clase

- Pachetul Model este alcatuit din 3 clase: Monomial, Polynomial si Operations.

Clasa Monomial este clasa fundamentala a aplicatiei ce defineste structura de monom. Aceasta clasa are ca si variabile instanta un coeficient si un grad. Am ales ca tipul coeficientului sa fie double deoarece, desi polinoamele introduse de utilizator vor avea coeficienti intregi, pot sa apara coeficienti reali la operatiile de impartire si integrare. Un coeficient de tip double poate fi transformat intr-un coeficient intreg printr-un cast de tip explicit, fara nicio problema. Tot in aceasta clasa se regasesc doua metode de afisare a monomului, singura diferenta dintre ele fiind tipul coeficientului care va fi afisat: metoda *printIntegerCoefficient* va afisa un coeficient intreg, iar metoda *printCoefficient* va afisa un coeficient real.

Clasa Polynomial - este clasa care sta la baza clasei Operations. Cum monomul este un polinom cu un singur termen, putem deduce ca polinomul este de fapt o expresie formata din mai multe monoame (termeni). In clasa aceasta am luat ca si variabila instanta o lista de monoame pentru a stoca lista de termeni ai polinomului introdus de utilizator, si fiecarui termen ii va corespunde un coeficient si un grad. Tot in aceasta clasa se regaseste metoda *constructMonomialsList* care are o importanta deosebita deoarece preia input-ul utilizatorului care este de fapt un String si il "traduce" intr-o lista de monoame. Aceasta conversie se realizeaza cu ajutorul expresiilor regulate dupa cum urmeaza:

-inlocuiesc toate '+' cu '+-'. Ex: $2x^3$-3x-1 va deveni $2x^3$+-3x+-1

-fac impartirea termenilor dupa '+'. Ex: [$2x^3$, -3x, -1]

-verific fiecare termen daca respecta unul dintre cele cazurile urmatoare:

Caz 1: termeni de forma *a (a-constanta)* ➡ "[+-]?\\d+"

Caz 2: termeni de forma *ax (de gradul I), a*$\in\mathbb{N}^*$ ➡ "[+-]?\\d*x"

Caz 3: termeni de forma ax^p *(de grad* $\geq$*1),* $a\in\mathbb{N}^*$, $p\in\mathbb{N}^*$ ➡ "[+-]?\\d*x\\^\\d+"

-in cazul in care unul din cazurile de mai sus nu e respectat, se genereaza o exceptie

Metoda *displayRealCoefficients* este utilizata in afisarea polinoamelor rezultat ale operatiilor de impartire si integrare, unde pot sa apara coeficienti reali.

O alta clasa importanta in construirea modelului este clasa ArrayListAnySize din pachetul Utils. Aceasta clasa este descendenta a clasei ArrayList, si are ca si caracteristica principala faptul ca intr-o instanta a unei astfel de clase, cum este cazul listei de monoame din clasa Polynomial se poate adauga un element la orice index, restul elementelor precedente indexului la care s-a adaugat elementul fiind initializate cu un element de referinta- in cazul nostru un monom cu coeficient 0 si grad 0, pentru prima adaugare.

Clasa Operations

Daca pana acum rolul claselor a fost sa defineasca operanzii cu care calculatorul de polinoame va lucra si anume polinoamele, rolul clasei Operations este de a implementa operatiile cu acesti operanzi. Astfel in aceasta clasa regasim 6 metode corespunzatoare a 6 operatii matematice: adunare, scadere, inmultire, impartire, derivare si integrare.

Pentru o mai buna intelegere voi descrie fiecare metoda in parte :

- Metoda *add (Polynomial, Polynomial)* –reprezinta operatia de adunare a doua polinoame. Aceasta se realizeaza prin adunarea coeficientilor corespunzatori

termenilor de acelasi grad. Polinoamele fiind construite printr-o lista in care gradele cresc odata cu indecsii, la indexul n avand un polinom de grad n cu coeficientul corespunzator, aceasta operatie este foarte usor de realizat.

$$P(X)+Q(X)=(a_n+b_n)*X^n+(a_{n-1}+b_{n-1})*X^{n-1}+\cdots+(a_1+b_1)*X+(a_0+b_0)$$

- Metoda *subtract (Polynomial, Polynomial)* – reprezinta operatia de scadere a doua polinoame. Aceasta se realizeaza prin scaderea din coeficientii primului polinom a coeficientilor de grad corespunzator din cel de-al doilea polinom.

$$P(X)-Q(X)=(a_n-b_n)*X^n+(a_{n-1}-b_{n-1})*X^{n-1}+\cdots+(a_1-b_1)*X+(a_0-b_0)$$

- Metoda *multiply (Polynomial, Polynomial)* – reprezinta operatia de inmultire a doua polinoame. Aceasta se realizeaza prin inmultirea fiecarui termen al primului polinom cu toti termenii celui de-al doilea si adunarea termenilor de acelasi grad din rezultat.

$P(X)=3*X^2-X+1$
$Q(X)=X-2$
$=> P(X)*Q(X)=3*X^3-X^2+X-6*X^2+2*X-2=3*X^3-7*X^2+3*X-2$

- Metoda *divide (Polynomial, Polynomial)* – reprezinta operatia de impartire a doua polinoame. Aceasta se realizeaza prin impartirea polinomului de grad mai mare la cel de grad mai mic sau egal cu al sau. Pentru a implementa aceasta operatie am utilizat urmatorul algoritm:

Polynomial long division [2]

```
function n / d is
    require d ≠ 0
    q ← 0
    r ← n             // At each step n = d × q + r

    while r ≠ 0 and degree(r) ≥ degree(d) do
        t ← lead(r) / lead(d)    // Divide the leading terms
        q ← q + t
        r ← r − t × d

    return (q, r)
```

where +, −, and × represent polynomial arithmetic

$P(X)=X^3-2*X^2+6*X-5$

$Q(X)=X^2-1$

$(X^3-2*X^2 + 6*X-5): (X^2-1) = X-2$

$-X^3 \quad + X$

$-2*X^2 + 7*X-5$

$2*X^2 \quad -2$

$7*X-7$

=>Quotient = X-2; Remainder = 7*X-7

- Metoda *derive (Polynomial)* – reprezinta operatia de derivare a unui polinom. Aceasta se realizeaza prin inmultirea coeficientului corespunzator fiecarui termen cu gradul sau si reducerea ordinului sau cu 1.

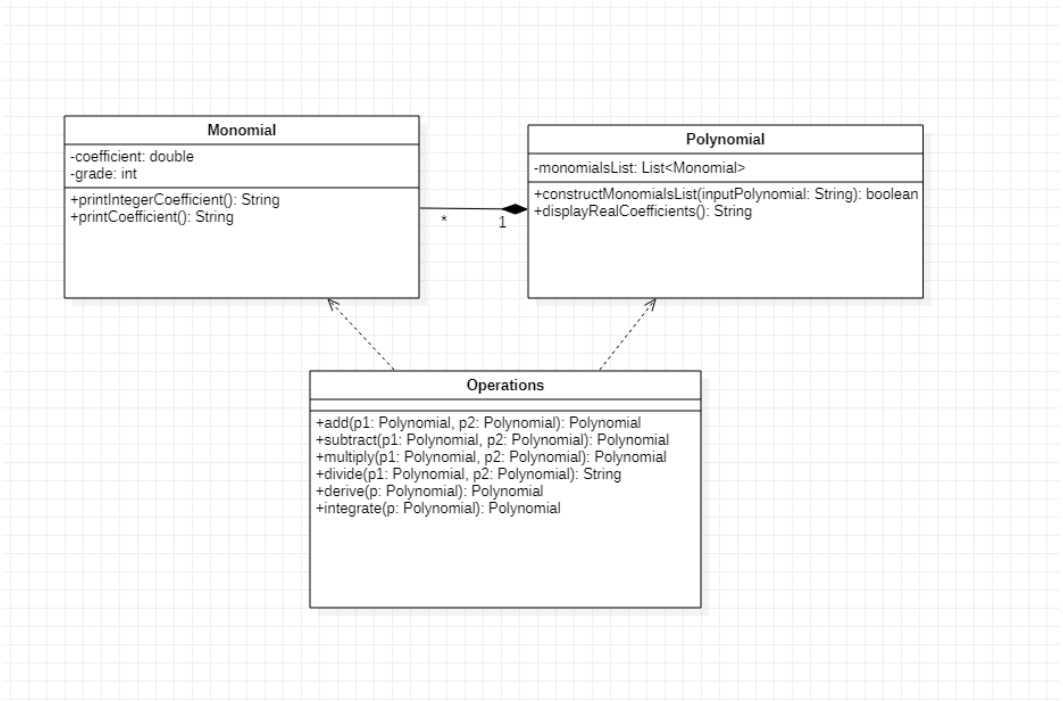
$$\frac{d}{dx} a_n*X^n+a_{n-1}*X^{n-1}+\cdots+a_1*X+a_0=n*a_n*X^{n-1}+(n-1)*a_{n-1}*X^{n-2}+\cdots+a_1$$

- Metoda *integrate (Polynomial)* – reprezinta operatia de integrare a unui polinom. Aceasta se realizeaza prin imparitrea coeficientului corespunzator fiecarui termen prin (gradul sau + 1) si cresterea ordinului cu 1.

$$\int a_nX^n + a_{n-1} * X^{n-1} + \cdots + a_1 * X + a_0 =$$

$$= \int a_nX^n dx + \int a_{n-1} * X^{n-1} dx + \cdots + \int a_1 * X\, dx + \int a_0 dx\,,$$

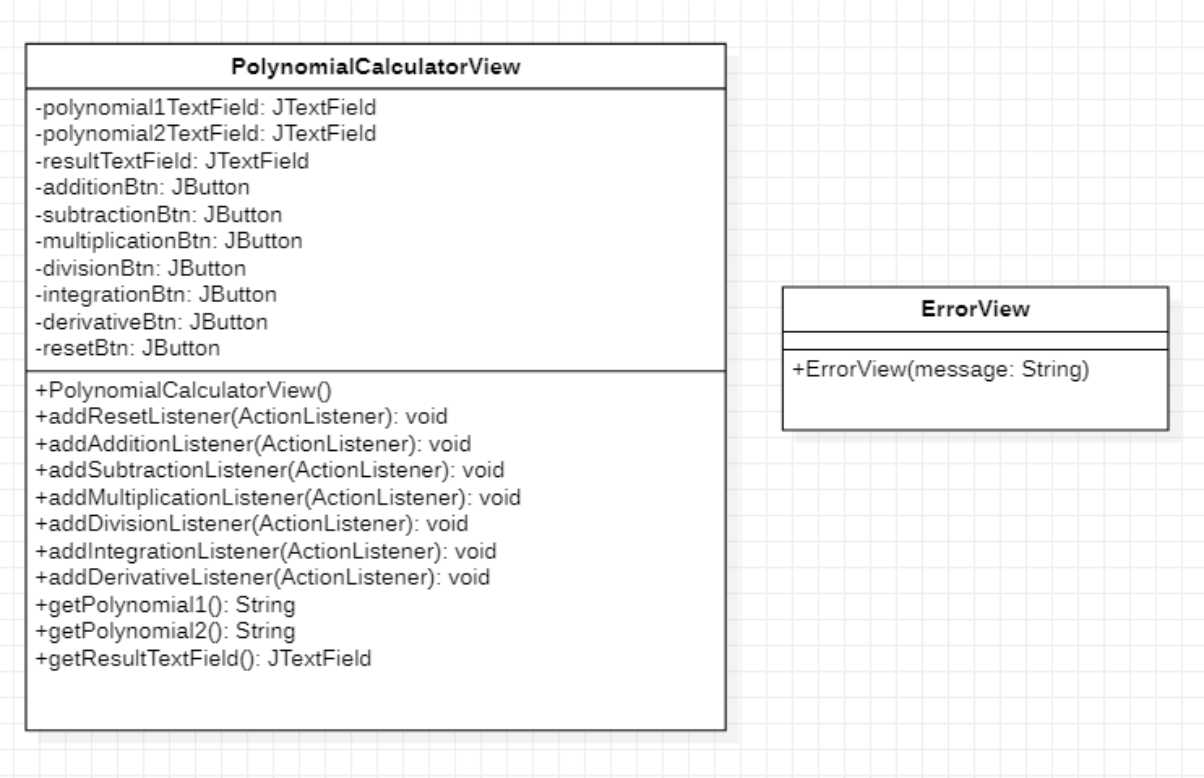
unde $\int a_n * X^n dx = \frac{a_n*X^{n+1}}{n+1} + C$



- Pachetul View este alcatuit din 2 clase: PolynomialCalculatorView si ErrorView. Clasele de aici sunt dezvoltate folosind pachetul Swing.

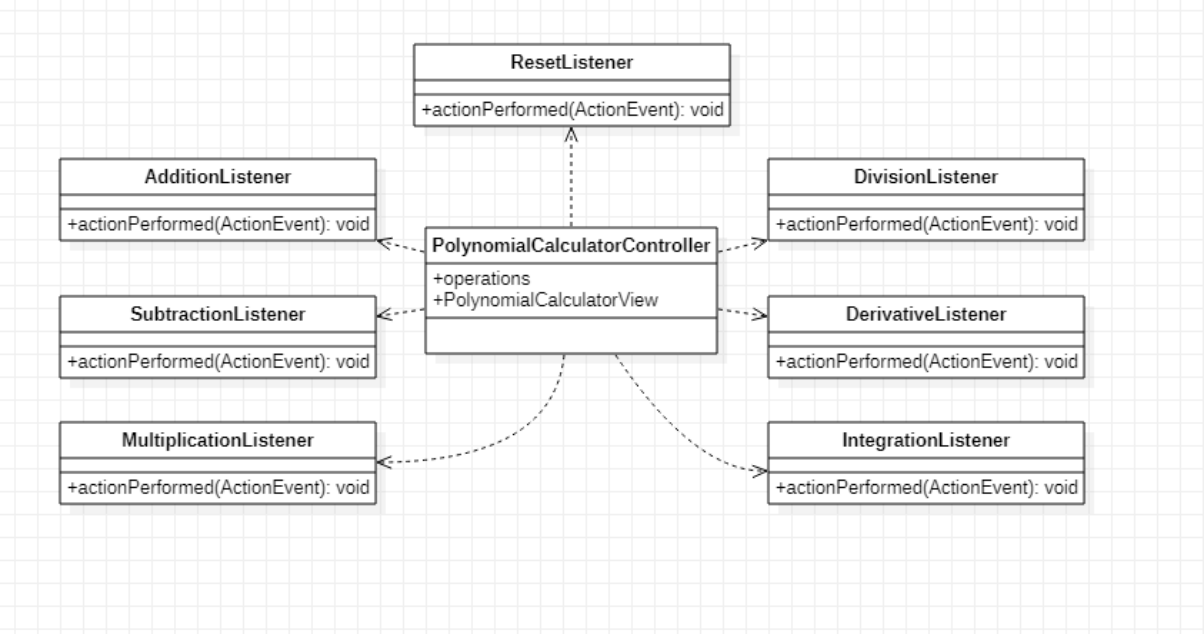
  Clasa PolynomialCalculatorView – contine interfata grafica a calculatorului. Aceasta din urma este compusa din 2 JTextField-uri unde se introduc cele doua polinoame, un JTextField pentru afisarea rezultatului, 6 butoane pentru selectia operatiei dorite + un buton de Reset care goleste campurile corespunzatoare celor 2 polinoame de intrare si cel al rezultatului.
  Toate componentele care alcatuiesc interfata grafica: butoane, panel-uri, label-uri, campuri de text sunt inglobate intr-un top-level container (contentPanel).

  Clasa ErrorView este o clasa din GUI special construita pentru afisarea mesajelor de eroare rezultate in urma interactiunii necorespunzatoare a utilizatorului cu aplicatia. Astfel de fiecare data cand un polinom este introdus gresit sau nu s-au introdus suficiente polinoame pentru a efectua o anumita operatie utilizatorul este avertizat prin deschiderea unei ferestre noi care contine un mesaj corespunzator.



- Pachetul Controller contine o singura clasa: PolynomialCalculatorController care este alcatuita din mai multe clase interne (ascultatori ai butoanelor din partea de view).

  Clasa PolynomialCalculatorController este una esentiala, intrucat aceasta face legatura intre model si view. Aici se preia input-ul utilizatorului, se verifica daca au fost introduse corect datele si daca se poate efectua operatia, in caz contrar generandu-se mesaje de eroare corespunzatoare. Daca operatia este posibila, atunci controllerul da comanda de efectuare a acesteia si seteaza rezultatul pentru a putea sa fie vizualizat de utilizator.



Implementare

- Pentru implementarea aplicatiei am folosit mediul de dezvoltare IntelliJ IDEA.

  Odata ce am analizat problema, am definit scenariile si cazurile de utilizare, structurile de date necesare pentru aplicatie cat si niste diagrame UML de pachete, respectiv de clase pentru a surprinde fiecare detaliu, se poate trece la pasul de implementare, de transpunere in cod a ideilor de proiectare. Pentru partea de implementare intervin caracteristicile limbajului folosit, in cazul nostru limbajul Java.

  Nu voi intra in foarte multe detalii, intrucat am incercat sa adaug cat mai multe comentarii in codul sursa pentru o intelegere cat mai buna.

➔ Implementarea metodei *subtract(Polynomial, Polynomial)* din **clasa Operations**

```java
public Polynomial subtract(Polynomial polynomial1, Polynomial polynomial2) {
    //Declaram polinomul result unde vom stoca rezultatul operatiei de scadere
    Polynomial result = new Polynomial();

    // maxSize - retine maximul gradelor celor 2 polinoame
    int maxSize = Math.max(polynomial1.getMonomialsList().size(), polynomial2.getMonomialsList().size());
    ListIterator<Monomial> it1 = polynomial1.getMonomialsList().listIterator(polynomial1.getMonomialsList().size());
    ListIterator<Monomial> it2 = polynomial2.getMonomialsList().listIterator(polynomial2.getMonomialsList().size());
    int index = maxSize - 1;

    //parcurgem listele de termeni ai polinoamelor de la gradul cel mai mare la cel mai mic
    while (it1.hasPrevious() || it2.hasPrevious()) {

        //daca polinomul 1 este de grad mai mare decat cel al 2-lea ii adaugam termenii in rezultat atata timp cat gradul sau este mai mare
        while (it1.previousIndex() > it2.previousIndex()) {
            result.getMonomialsList().add(index, new Monomial(it1.previous().getCoefficient(), index));
            index--;
        }
        //daca polinomul 2 este de grad mai mare decat primul ii adaugam termenii in rezultat atata timp cat gradul sau este mai mare, dar cu semne contrare
        while (it2.previousIndex() > it1.previousIndex()) {
            result.getMonomialsList().add(index, new Monomial( coefficient: (-1) * it2.previous().getCoefficient(), index));
            index--;
        }

        //cand polinoamele au ajuns de acelasi grad, scadem coeficientii corespunzatori din fiecare polinom si ii adaugam la rezultat
        result.getMonomialsList().add(index, new Monomial( coefficient: it1.previous().getCoefficient() - it2.previous().getCoefficient(), index));
        index--;

    }
    return result;
}
```

Operatia de adunare se realizeaza similar.

➔ **Clasa ArrayListAnySize :**

```java
public class ArrayListAnySize<E> extends ArrayList<E>{
    private E referenceElem;

    public E getReferenceElem() { return referenceElem; }

    public void setReferenceElem(E referenceElem) { this.referenceElem = referenceElem; }

    public void add(int index, E element){

        if(index >=0 && index <= size()){
            if(size()!=0)
            super.remove(index);
            super.add(index, element);
            return;
        }

        int insertZeros = index - size();
        for(int i = 0; i < insertZeros; i++){ //elem to insert in between the elements of the array
            super.add(referenceElem);
        }
        super.add(element);
    }
}
```

Dupa cum am precizat si in partea de Proiectare, aceasta clasa este foarte importanta deoarece ne ajuta sa adaugam un element intr-un array la orice index. Cum utilizatorul nostru introduce polinoamele in ordine descrescatoare a gradelor, in lista de monoame le vom adauga tot in aceasta ordine. Astfel pentru un polinom de grad n primul termen adaugat in lista de monoame va fi coeficientul termenului de grad n, impreuna cu gradul, adaugat la indexul corespunzator gradului. Am decis ca indexul sa corespunda cu gradul termenului pentru a simplifica operatiile si parcurgerile pe liste. Deci la adaugarea primului termen, cel de grad maxim, toti termenii precedenti vor fi initializati cu un element de referinta -in cazul acesta am ales sa fie un monom cu coeficient 0 si grad 0. La fiecare noua adaugare de dupa acest pas, va fi sters termenul existent la acel index (elemental de referinta) si se va adauga elementul dorit.

➔ Implementarea metodei *displayRealCoefficients()* din **clasa Polynomial**

```
public String displayRealCoefficients(){
    String str="";
    // parcurgem lista termenilor de la sfarsit (vrem sa afisam gradele descrescator)
    ListIterator<Monomial> it = monomialsList.listIterator(this.monomialsList.size());
    while (it.hasPrevious()) {
        Monomial prev=it.previous();
        // pentru fiecare termen apelam metoda printCoefficient din clasa Monom
        str+=prev.printCoefficient();

    }
    return str;
}
```

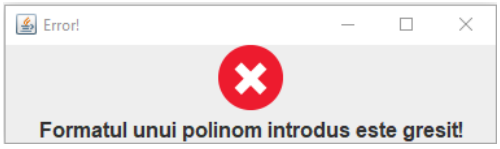
➔ Implementarea metodei *printCoefficient()* din **clasa Monomial**

```
public String printCoefficient() {
    String str="";
    if(coefficient>0 ) str+="+" +String.format("%.2f",this.coefficient);
    if(coefficient<0 ) str+="-"+ String.format("%.2f",Math.abs(this.coefficient));

    if(coefficient!=0) {
        if (grade > 1) str += "x^" + this.getGrade();
        if (grade == 1) str += "x";
    }
    return str;
}
```

➔ Implementarea **clasei ErrorView**

```
package view;

import javax.swing.*;
import java.awt.*;

public class ErrorView extends JFrame {
    public ErrorView(String message){
        JLabel errorLabel=new JLabel(message,new ImageIcon( filename: "src/utils/icons/errorIcon.png"),JLabel.CENTER);
        errorLabel.setVerticalTextPosition(JLabel.BOTTOM);
        errorLabel.setHorizontalTextPosition(JLabel.CENTER);
        errorLabel.setFont(new Font( name: "OpenSans",Font.BOLD, size: 16));
        this.add(errorLabel);
        this.setTitle("Error!");
        this.pack();
        this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        this.setLocationRelativeTo(null);
        this.setVisible(true);

    }
}
```

Exemple de erori:

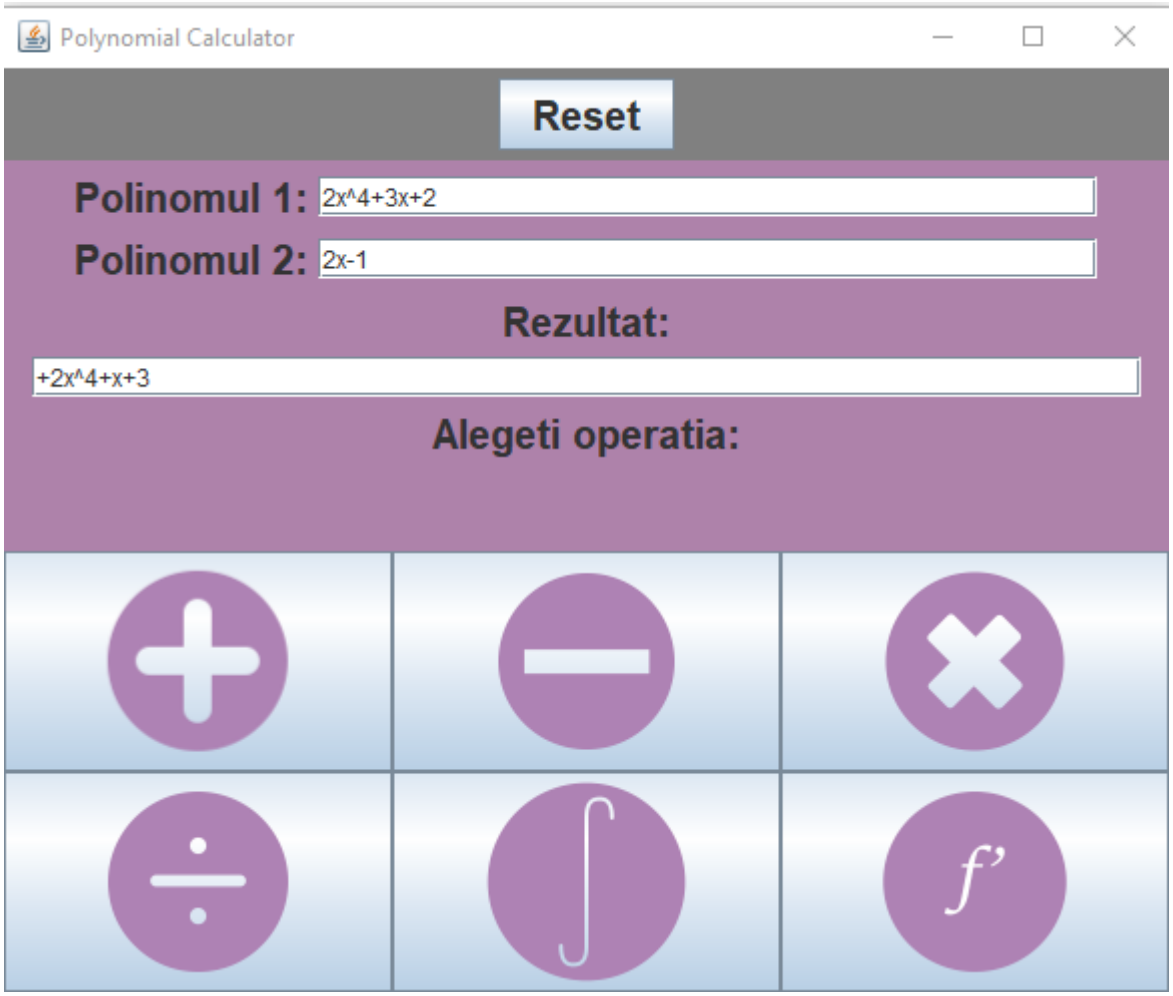


➔ Implementarea ascultatorului de scadere din **clasa PolynomicalCalculatorController**

```
class SubtractionListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {
        Polynomial polynomial1 = new Polynomial();
        Polynomial polynomial2 = new Polynomial();
        try {
            if (polynomialCalculatorView.getPolynomial1().isEmpty() || polynomialCalculatorView.getPolynomial2().isEmpty()) {
                new ErrorView("Trebuie sa introduceti 2 polinoame pentru a efectua operatia!");
            } else {

                if (!polynomial1.constructMonomialsList(polynomialCalculatorView.getPolynomial1()))
                    new ErrorView("Primul polinom este introdus gresit! Introduceti din nou acest polinom!");
                else if (!polynomial2.constructMonomialsList(polynomialCalculatorView.getPolynomial2()))
                    new ErrorView("Al doilea polinom este introdus gresit! Introduceti din nou acest polinom!");

                else {

                    Polynomial result = operations.subtract(polynomial1, polynomial2);
                    if (result.toString().isEmpty())
                        polynomialCalculatorView.getResultTextField().setText("0");
                    else
                        polynomialCalculatorView.getResultTextField().setText(result.toString());
                }
            }
        } catch (Exception ex) {
            new ErrorView("Formatul unui polinom introdus este gresit!");
        }
    }
}
```

Rezultate

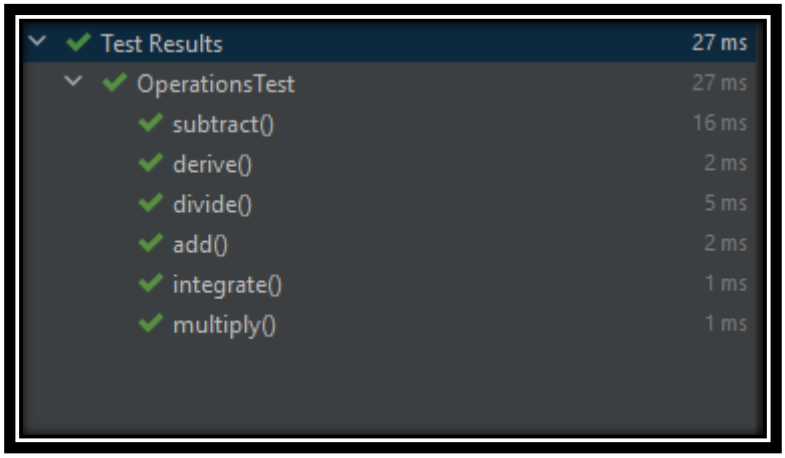
➔ Rezultatul operatiei de scadere



Pentru partea de testare am folosit framework-ul JUnit. Am testat fiecare metoda din **clasa Operations** pentru a ma asigura ca functionalitatea este cea dorita. Rezultatele au fost cele asteptate, in cazul in care polinoamele introduse au fost corecte. Am reusit sa tratez si cazurile in care pot sa apara erori in introducerea polinoamelor si astfel rezultatele sa nu fie cele dorite. Metoda *constructMonomialsList(inputPolynomial: String)* returneaza o valoare booleana, true daca fiecare termen al polinomului a respectat unul dintre cele trei pattern-uri descriese in etapa de proiectare, false in caz contrar.

De asemenea inainte de a integra fiecare componenta in intreg, am incercat sa testez functionalitatea ei, definindu-mi input-uri specifice pentru rolul pe care aceasta ar trebui sa-l aiba in aplicatie.

Mai jos sunt prezentate rezultatele testatii operatiilor folosind JUnit si se poate observa cum testele au avut success, adica rezultatul asteptat corespunde cu cel furnizat de operatie.




```java
@Test
void subtract() {
    operation="Subtraction";

    try{
    polynomial1.constructMonomialsList( inputPolynomial: "6x-3");
    polynomial2.constructMonomialsList( inputPolynomial: "2x-1");
    expectedResult="+4x-2";
    }
    catch (Exception e)
    {
        System.out.println(e.getMessage());
    }
    assertEquals(expectedResult,operations.subtract(polynomial1,polynomial2).toString(), message: "Subtraction Operation-Failed!");
}
```

Concluzii

Desi nu a fost un parcurs usor in dezvoltarea acestei aplicatii, si m-am lovit de mai multe obstacole, ma bucur ca in final am reusit sa le depasesc si am invatat din ele.

Daca la inceput totul mi s-a parut relative greu, pe masura ce am avansat in dezvoltarea proiectului mi-am dat seama ca pentru toate este o solutie.

Ce consider ca am invatat din acest proiect, este faptul ca este necesara o analiza foarte atenta a cerintelor, o proiectare la fel de atenta si ca trebuie luate toate cazurile in considerare pentru a definii cat mai corect modelul de date. Preluarea input-ului utilizatorului a fost iarasi o provocare, care a necesitat o munca destul de intensa, pentru a lua in considerare toate neajunsurile care pot sa apara.

Si cum in orice este loc de mai bine, si aplicatia aceasta ar putea fi imbunatatita si extinsa. De la design-ul interfetei grafice pana la adaugarea de noi functionalitati, aplicatia poate fi imbunatatita in asa fel incat operatiile cu polinoame sa se realizeze cat mai interactiv si sa ne faca viata mai usoara, intrucat avem multe aplicatii ale acestora in viata cotidiana.

Bibliografie

[1]- https://ro.wikipedia.org/wiki/Polinom

[2]- https://en.wikipedia.org/wiki/Polynomial_long_division

[3]- FUNDAMENTAL PROGRAMMING TECHNIQUES (ASSIGNMENT 1 SUPPORT PRESENTATION)