MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
CATEDRA CALCULATOARE

Tehnici de programare

Documentație Proiect

Calculator de polinoame

Bîrle Silviu-Adrian

Grupa 30222

Cuprins

1. Obiectivul temei

Ca si obiectiv am avut de realizat un calculator pentru polinoame. Operațiile implementate sunt : adunarea , scăderea , înmulțirea , împărțirea , derivarea și integrarea. Cu ajutorul unei interfețe grafice care este user-friendly , utilizatorul poate introduce polinoamele și își poate alege o operație pe care calculatorul o va efectua. Rezultatul va fi afișat într-un chenar tot în interfața grafică.

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

❖ Analiza:

Ca să simplificăm problema am împărțit polinomul până am ajuns la un nivel atomic , adică din Polinom → Monom → Exponent și Coeficient. Astfel că , având coeficienții și exponenții putem crea monoame care la rândul lor vor construi polinoame.

Pentru implementarea operațiilor am folosit diferiți algoritmi.

❖ Modelare:

În primul rând trebuie să transformăm input-ul utilizatorului care este de tip String într-un polinom. Pentru a putea lucra ușor cu el , folosind un regex despărțim String-ul în coeficienți și exponenți care vor forma Monoame , care vor alcătui Polinoame. Ulterior , vom putea folosi operațiile implementate pe polinoame.

❖ Scenarii:

Pentru a nu exista niciun fel de eroare , utilizatorul trebuie să respecte următoarele condiții :

- Necunoscuta polinoamelor este <x> sau <X> .
- Termenii vor fi delimitați numai prin semnele `+` și `-`
- Fiecare monom va fi unul din următoarele :
  - [+-] <coeficient>**X**^<exponent>
  - [+-]<coeficient>**X**
  - [+-]**X**^<exponent>
  - [+-]**X**

Alte observații :

- Dacă input-ul pentru un polinom conține doar litere , se va afișa un mesaj de eroare.

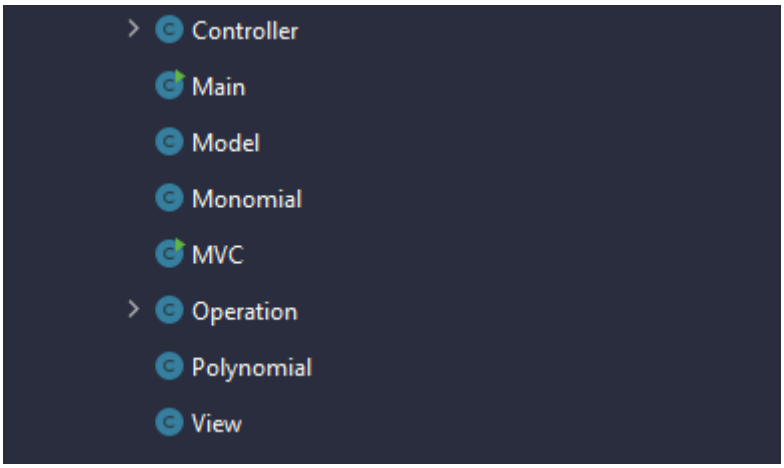
- Dacă nu se va introduce un polinom și se cere execuția unei operații , se va afișa un mesaj de eroare.
- Este permis spațiul între caractere
- Nu este obligatoriu ca primul termen să aibă `+` în față dacă este pozitiv.

❖ Cazuri de utilizare :

După apăsarea butonului **RUN** se deschide interfața grafică care este user-friendly , conține 6 butoane fiecare reprezentând o operație , 3 chenare text unde se vor introduce polinoamele (primele două pentru două polinoame care vor fi folosite pentru una din operațiile : Adunare , Scădere , Înmulțire și Împărțire , iar ultimul pentru a introduce un polinom care va efectua una din operațiile : derivare sau integrare) și un alt chenar text în care nu se poate introduce text , care constituie Rezultatul operației efectuate.
După efectuarea unei operații polinoamele nu se vor șterge si vor putea fi utilizate pentru alte operații.

3. Proiectarea (decizii de proiectare , diagrama UML , structuri de date , proiectare clase , interfete , relatii , packages , algoritmi ,interfață utilizator)

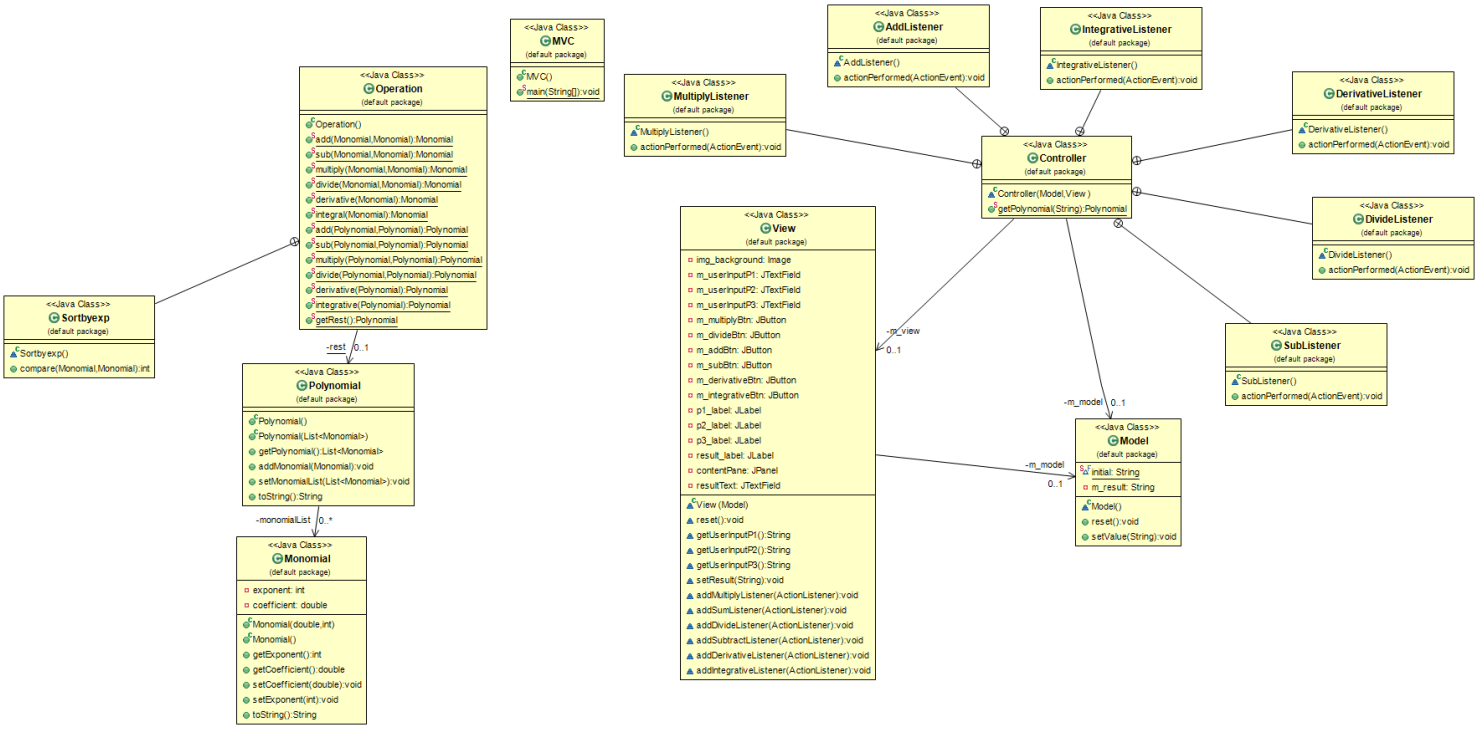
Programul a fost împărțit în mai multe clase pentru a fi mai ușor de înțeles:



Clasele **MVC** , **Controller** , **Model** și **View** reprezintă un pattern arhitectural
Clasa **Main** conține niște polinoame care operațiile implementate pentru a fi testate în consolă
Clasa **Operation** conține implementarea tuturor operațiilor (Adunare ,

Scădere , Înmulțire , Împărțire , Derivare și Integrare)
Clasa **Monomial** conține implementarea unui monom
Clasa **Polynomial** conține implementarea unui polinom

Diagrama UML :



Interfață Grafică :

- **Frame** – fereastra care se deschide atunci când rulăm programul și care conține toate elementele legate de partea de interfață grafică a proiectului. Precum majoritatea aplicațiilor are în partea dreaptă sus 3 butoane : `--` Restore/Punerea in bară , `patratul` e minimizarea /maximizarea ferestrei si `X` reprezintă închiderea programului, dar minimizarea/maximizarea a fost dezactivată.
- **TextField-uri** – chenarele unde utilizatorul introduce polinoamele de la tastatură sub formă de String-uri. Avem 4 astfel de textfield-uri : 3 pentru a introduce câte un polinom și unul în care nu se poate scrie reprezentând rezultatul operațiilor efectuate.
- **JLabel-uri** – etichete care conțin mesaje care nu se pot modifica de către utilizator.
  - **First Polynomial** - se află deasupra textfield-ului unde se introduce

primul polinom

- **Second Polynomial** - se află deasupra textfield-ului unde se introduce al doilea polinom
- **Third Polynomial** - se află deasupra textfield-ului unde se introduce al treilea polinom
- Un label care are dimensiunea frame-ului si conține imaginea de background

- **Butoane** - există 6 butoane , câte unul pentru fiecare operație
  - **Add** – face suma a două polinoame
  - **Subtract** – face scăderea a două polinoame
  - **Multiply** – face înmulțirea a două polinoame
  - **Divide** – face împărțirea a două polinoame
  - **Derivative** – face derivata unui polinom
  - **Integrative** – face integrala unui polinom

4. Implementare

Au fost implementate 6 operații : adunarea adouă polinoame , scăderea a două polinoame , înmulțirea a două polinoame , împărțirea a două polinoame , derivarea unui polinom si integrarea unui polinom .

Programul structurat pe clase :

- Clasa **MVC** – aici se face instanțierea pattern-ului Arhitectural numit și MVC (Model View Controller) si se face rularea aplicației.

```
public class MVC {
    public static void main(String[] args) throws IOException {

        Model       model       = new Model();
        View        view        = new View(model);
        Controller  controller  = new Controller(model,view);

        view.setVisible(true);
    }
}
```

- Clasa **Model** – conține date precum instanțierea textfield-ului care inițiază rezultatul cu `` . (String gol)

```
public class Model {
    static final String initial="";
    private String m_result;
    Model(){
        reset();
    }
    public void reset() {
        m_result = initial;
    }

    public void setValue(String value) {
        m_result = value;
```

- Clasa **View** – aici se află partea de design a interfeței grafice și conține : Frame-ul , Butoanele , Label-urile și TextField-urile.

```
private JTextField m_userInputP1 = new JTextField();
private JTextField m_userInputP2 = new JTextField();
private JTextField m_userInputP3 = new JTextField();
private JButton m_multiplyBtn = new JButton( text: "Multiply");
private JButton m_divideBtn = new JButton( text: "Divide");
private JButton m_addBtn = new JButton( text: "Add");
private JButton m_subBtn = new JButton( text: "Subtract");
private JButton m_derivativeBtn = new JButton( text: "Derivative");
private JButton m_integrativeBtn = new JButton( text: "Integrative");
private JLabel p1_label = new JLabel( text: "First Polynomial 1");
private JLabel p2_label = new JLabel( text: "Second Polynomial 2");
private JLabel p3_label = new JLabel( text: "Third Polynomial 3");
private JLabel result_label = new JLabel( text: "Result");
private JPanel contentPane;
private JTextField resultText;

private Model m_model;
```

- Clasa **Controller** – conține partea funcțională a interfeței grafice : extragerea String-urilor din textfield-uri , Listenerii pentru butoane , afișarea rezultatului după efectuarea unei operații.

```
Controller(Model model, View view) {
    m_model = model;
    m_view = view;

    view.addSumListener(new AddListener());
    view.addSubtractListener(new SubListener());
    view.addMultiplyListener(new MultiplyListener());
    view.addDivideListener(new DivideListener());
    view.addDerivativeListener(new DerivativeListener());
    view.addIntegrativeListener(new IntegrativeListener());
}

class AddListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        String userInputP1 = "";
        String userInputP2 = "";
        userInputP1 = m_view.getUserInputP1();
```

- Clasa **Main** – conține niște polinoame de test care efectuează toate operațiile implementate și afișează rezultatele în consolă.

```
Polinomul 1: 6x^4+3x^3+2x^2+2x
Polinomul 2: 2x^3+5x^2+3x
-------Suma polinoamelor-----
6x^4+5x^3+7x^2+5x
-------Diferenta polinoamelor------
6x^4+x^3-3x^2-x
-------Inmultirea polinoamelor------
12x^7+36x^6+37x^5+23x^4+16x^3+6x^2
-------Impartirea polinoamelor------
Catul este: 3x-6
Restul este: 23x^2+20x
-------Derivata unui polinom------
24x^3+9x^2+4x+2
-------Integrala unui polinom------
1.2x^5+0.75x^4+0.6666666666666666x^3+x^2
```

- Clasa **Monomial** – conține atributele monomului (coeficient și exponent) , Gettere și Settere pentru atributele monomului și un Override al metodei toString pentru afișarea unui monom în funcție de anumite reguli (de exemplu pentru coeficient = 1 , el nu se va mai afișa , în loc de 1x^2 se va afișa x^2)

```
public class Monomial {

    private int exponent;
    private double coefficient;

    public Monomial(double coefficient, int exponent)
    {
        this.coefficient = coefficient;
        this.exponent = exponent;
    }

    public Monomial(){

    }
    public int getExponent() { return exponent; }

    public double getCoefficient() { return coefficient; }

    public void setCoefficient(double coefficient) { this.coefficient = coefficient; }

    public void setExponent(int exponent) { this.exponent = exponent; }

    @Override
    public String toString(){
        if(this.coefficient==0){
            return "";
        } else if(this.exponent==0) {
            if(this.coefficient==Math.ceil(this.coefficient)) {return String.valueOf((int)this.coefficient);}
            else {return String.valueOf(this.coefficient);}
```

- Clasa **Polynomial** – conține atributele unui polinom (o listă de monoame) , gettere și settere și un Override la metoda toString pentru afișarea unui polinom.

```
private List<Monomial> monomialList;

public Polynomial(){
    monomialList = new ArrayList<>();
}
public Polynomial(List<Monomial> monomial) { monomialList = monomial; }
public List<Monomial> getPolynomial() {
    return monomialList;
}

public void addMonomial(Monomial m) {
    this.monomialList.add(m);
}

public void setMonomialList(List<Monomial> monomialList) { this.monomialList = monomialList; }

@Override
public String toString(){
    StringBuilder sb = new StringBuilder();
    for(Monomial el : this.monomialList){
```

- Clasa **Operation** – unde sunt implementate toate operațiile pentru monoame și pentru polinoame. Operațiile pe polinoame utilizează operațiile pe monoame (în cazul adunării pe monom și scăderii pe monom ele se utilizează știind că avem exponenții egali). Metodele pe monoame returnează un monom , iar metodele pe polinoame returnează un polinom
  - Adunarea a două monoame – se adună coeficienții

```
public static Monomial add(Monomial m1, Monomial m2) {
    double aux;
    aux = m1.getCoefficient() + m2.getCoefficient();
    Monomial m = new Monomial(aux,m1.getExponent());
    return m;
}
```

  - Scăderea a două monoame – se scad coeficienții

```
public static Monomial sub(Monomial m1, Monomial m2){
    double aux;
    aux = m1.getCoefficient() - m2.getCoefficient();
    Monomial m = new Monomial(aux, m1.getExponent());
    return m;
}
```

  - Înmulțirea a două monoame – se înmulțesc coeficienții și se adună exponenții

```
public static Monomial multiply(Monomial m1 , Monomial m2){
    int auxExp;
    double auxCoef;
    auxCoef = m1.getCoefficient() * m2.getCoefficient();
    auxExp = m1.getExponent() + m2.getExponent();
    Monomial m = new Monomial(auxCoef,auxExp);
    return m;
}
```

- Împărțirea a două monoame – se împart coeficienții și se scad exponenții

```
public static Monomial divide(Monomial m1, Monomial m2){
    int auxExp;
    double auxCoef;
    auxCoef = m1.getCoefficient() / m2.getCoefficient();
    auxExp = m1.getExponent() - m2.getExponent();
    Monomial m = new Monomial(auxCoef,auxExp);
    return m;
}
```

- Derivata unui monom – coeficientul devine coeficient*exponent și exponentul scade cu o unitate

```
public static Monomial derivative(Monomial m1){
    int exp;
    double coef;
    coef = m1.getCoefficient() * m1.getExponent();
    exp = m1.getExponent() - 1;

    Monomial m = new Monomial(coef,exp);
    return m;
}
```

- Integrala unui monom – coeficientul devine coef/(exp+1) , iar exponentul crește cu o unitate

```
public static Monomial integral(Monomial m1){
    int exp;
    double coef;
    coef = m1.getCoefficient()/(m1.getExponent()+1);
    exp = m1.getExponent() + 1;

    Monomial m = new Monomial(coef,exp);
    return m;
}
```

- Pentru **adunarea** a două polinoame , se introduce fiecare monom al fiecărui polinom într-un polinom nou – result – și se verifica fiecare monom din primul polinom cu fiecare monom al celui de-al doilea polinom dacă au acelasi exponent , iar dacă da , se apelează adunarea pe monom între cele două monoame , după se șterg din polinomul result și se adaugă monomul nou in polinomul result.
- Pentru **scăderea** a două polinoame procedeul e similar cu cel al adunării doar că se apelează metoda de scădere pe monom în cazul monoamelor cu exponenți egali.
- Pentru **înmulțirea** a două polinoame , se înmulțește fiecare monom din polinomul 1 cu fiecare monom din polinomul 2 (folosind metoda de înmulțire pe monom) și se adaugă monomul rezultat în polinomul – result – și după se verifică dacă în polinomul – result – avem termeni care au același exponent , iar dacă da , se face adunarea lor .
- Pentru **împărțire** am folosit algoritmul de împărțire a două polinoame care se folosește și în matematică și obțin 2 polinoame , unul care reprezintă câtul și unul care reprezintă restul. Pentru rest există creat un getter.
- Pentru **derivarea** unui polinom , în timpul parcurgerii fiecărui monom al unui polinom , se apelează metoda de derivare pe monom , iar monomul rezultat se adaugă in polinomul – result –
- Pentru **integrarea** unui polinom în timpul parcurgerii fiecărui monom al unui polinom , se apelează metoda de integrare pe monom , iar monomul rezultat se adaugă in polinomul – result –
- Înauntrul clasei Operation se mai află un static class **Sortbyexp** care implementează **Comparator** și sortează polinomul descrescător după exponent.

- În clasa **Controller** am implementat o metodă **getPolynomial()** care primește ca parametru un string reprezentând polinomul și folosind un **Regex** (**Pattern**) despart polinomul în monoame urmând a crea polinoame de tipul Polynomial nu String ca să putem apela metodele pe polinoame. Tot aici am implementat o metodă **onlyLetters()** care verifică daca textul introdus conține doar litere și nu reprezintă un polinom (se exclude cazul în care string-ul este `x` sau `X` care reprezintă polinomul 1x^1).

- Metoda de extragere a polinomului :

```
public static Polynomial getPolynomial(String p1) {
    String monomialPattern = "([-+]?)(\\d*\\.?\\d*)?([xX](\\^-?\\d*\\.?\\d*)?)?";
    Pattern pattern = Pattern.compile(monomialPattern);
    String pol1 = p1;
    pol1 = pol1.replaceAll( regex: "\\s+",  replacement: "");
    Matcher matcher = pattern.matcher(pol1);
    Monomial m;
    Polynomial p = new Polynomial();
    while (matcher.find()) {
        String g1 = matcher.group(1);
        String g2 = matcher.group(2);
        String g3 = matcher.group(3);
        String g4 = matcher.group(4);
        int coeff, exp;
        try {
            if (g1.isEmpty() && g2.isEmpty() && g3.equals(null) && g4.equals(null))
                break;
            else {
```

Regex-ul folosit împarte fiecare monom în 4 grupuri :

a) Semnul
b) Coeficientul
c) Termenul x cu exponentul
d) Exponentul

- Metoda de verificare a inputului dacă conține doar litere :

```
public static boolean onlyLetters(String text){
    int cnt = 0;
    if(text.isEmpty())
        return false;
    char[] chars = text.toCharArray();
    for(char c : chars){
        if(!Character.isLetter(c))
            return false;
        else
```

➢ Mai există 4 clase de **Test** unde avem **assert**-urile :

- **ControllerTest –** testez metoda de extragere a unui polinom și cea de verificare a inputului
- **MonomialTest** – testez metodele din clasa Monomial
- **OperationTest –** testez toate operațiile
- **PolynomialTest –** testez metodele din clasa Polynomial

```java
class ControllerTest {

    @Test
    void getPolynomial() {
        String polynomial = "2x^2+1";
        Polynomial p = new Polynomial();
        p = Controller.getPolynomial(polynomial);
        assertEquals( expected: "2x^2+1",p.toString());
    }
```

```java
class PolynomialTest {

    @Test
    void getPolynomial() {
        Monomial m = new Monomial( coefficient: 2, exponent: 2);
        Monomial m1 = new Monomial( coefficient: 3, exponent: 4);
        Polynomial p = new Polynomial();
        p.addMonomial(m);
        p.addMonomial(m1);
        assertEquals( expected: 2,p.getPolynomial().get(0).getCoefficient());
        assertEquals( expected: 2,p.getPolynomial().get(0).getExponent());
        assertEquals( expected: 3,p.getPolynomial().get(1).getCoefficient());
        assertEquals( expected: 4,p.getPolynomial().get(1).getExponent());
    }
```

```java
class OperationTest {

    @Test
    void add() {
        Monomial m = new Monomial( coefficient: 1,  exponent: 3);
        Monomial m1 = new Monomial( coefficient: 2,  exponent: 3);
        Monomial res = new Monomial();
        res = Operation.add(m, m1);
        assertEquals( expected: 3, res.getCoefficient());
        assertEquals( expected: 3, res.getExponent());
    }
```

```java
class MonomialTest {

    @Test
    void getExponent() {
        Monomial m = new Monomial( coefficient: 2, exponent: 2);
        assertEquals( expected: 2,m.getExponent());
    }
```

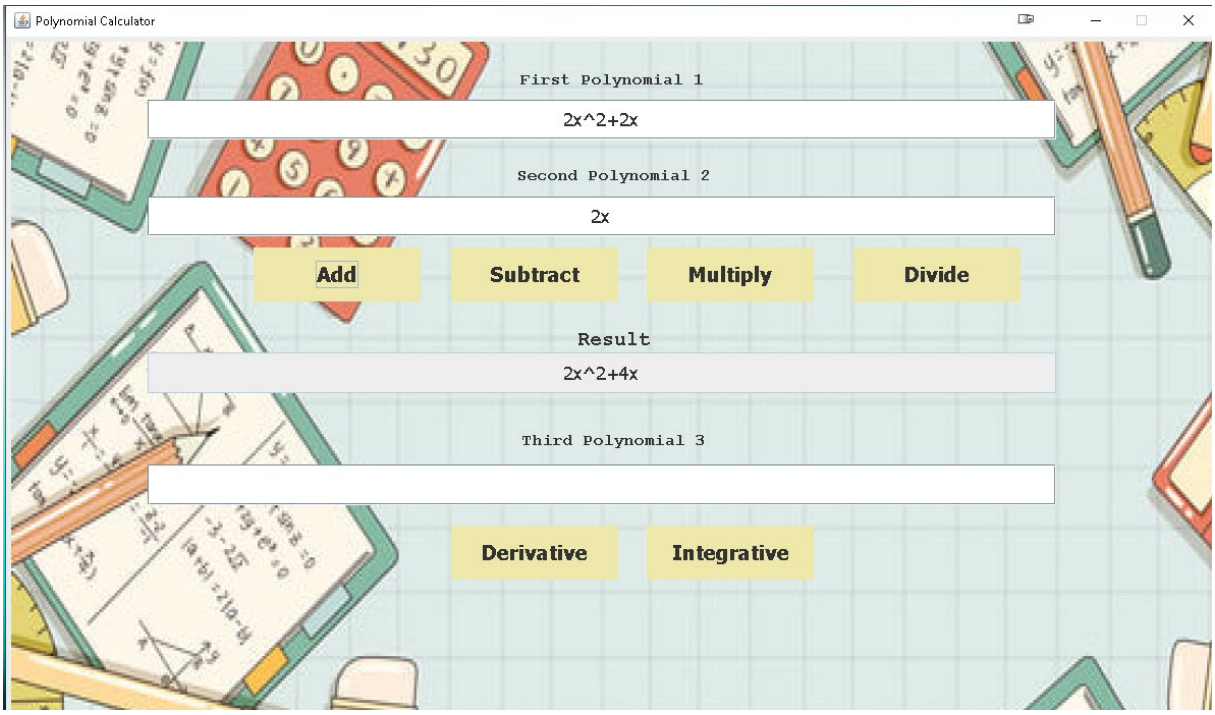
5. Rezultate

Pentru a arăta funcționalitatea fiecărei operații vom lua următoarele polinoame :

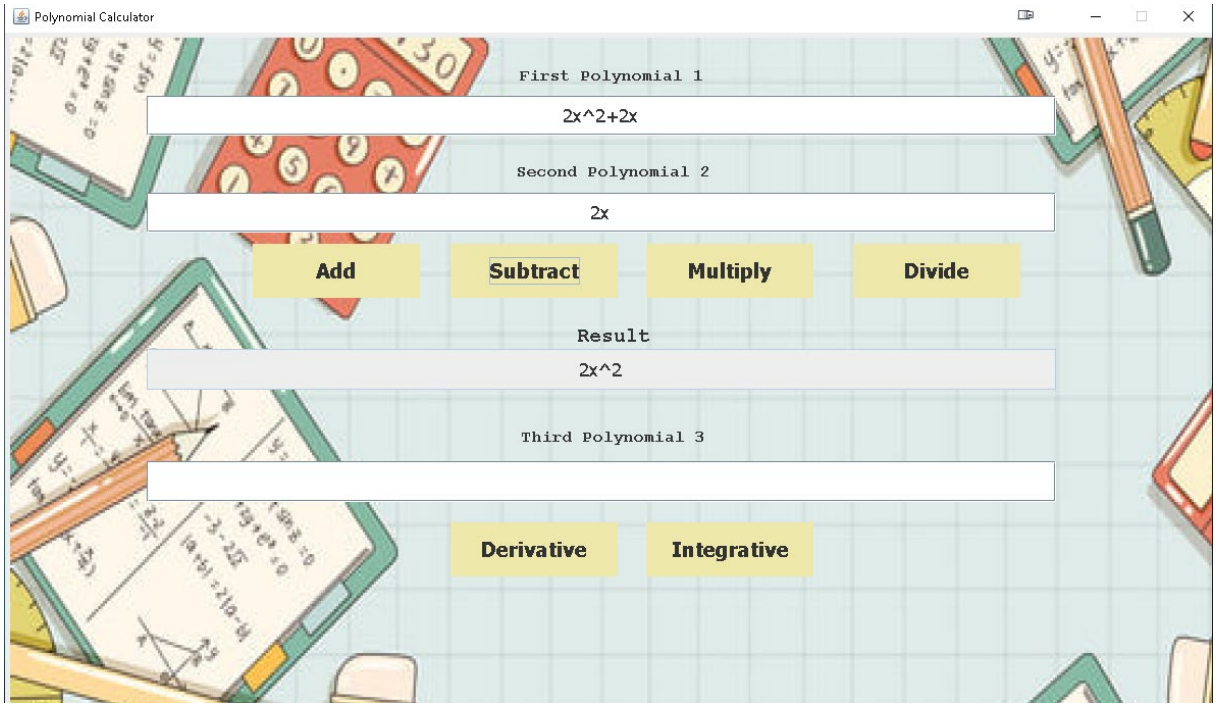
- First polynomial : 2x^2 + 2x
- Second polynomial : 2x
- Third polynomial : 4x^3

Testare

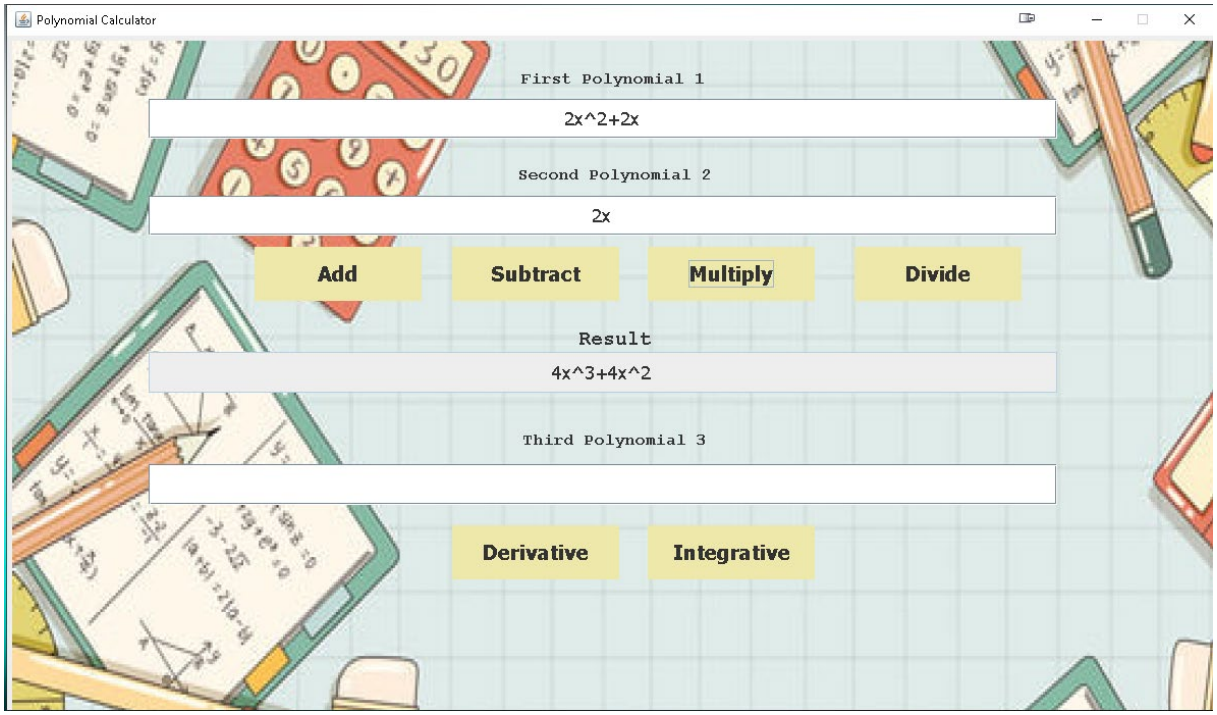
❖ Adunare



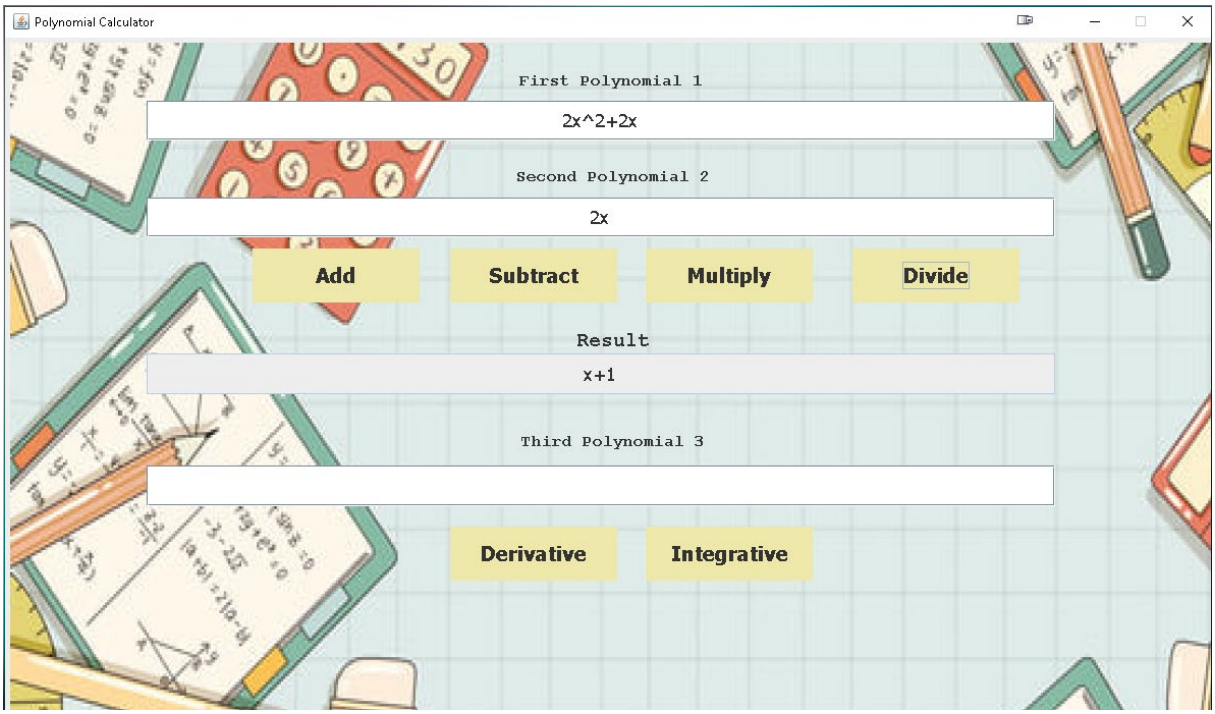
❖ Scădere



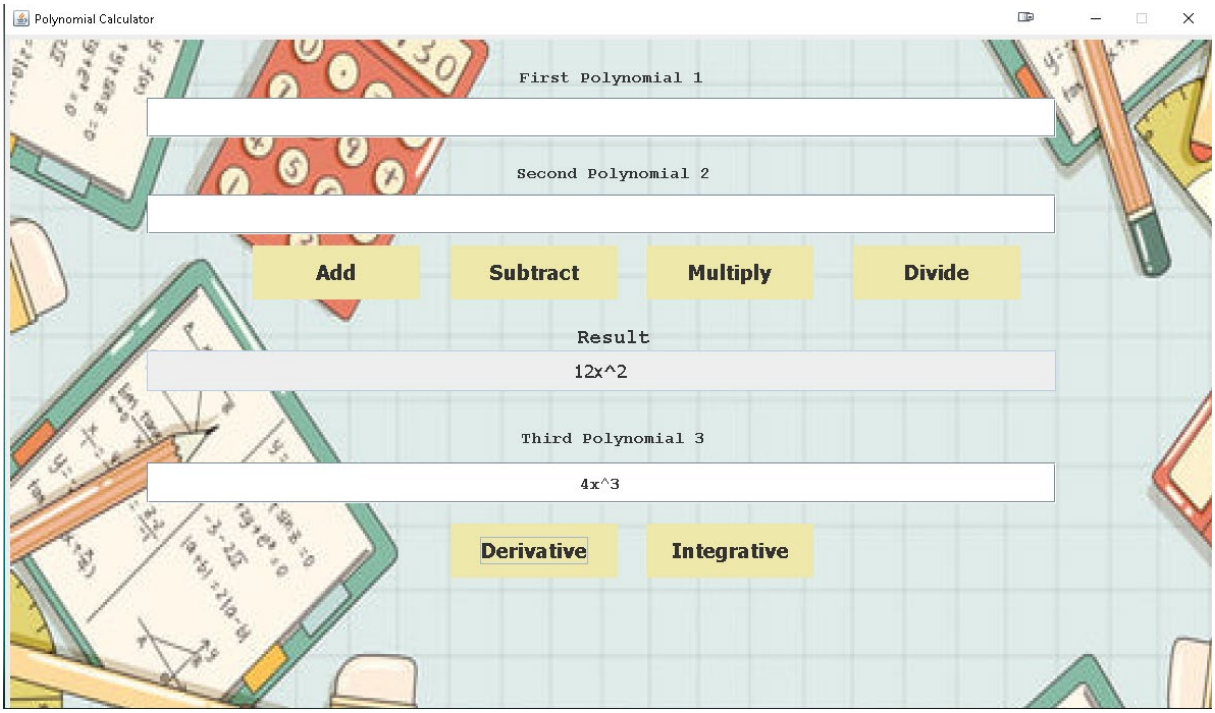
❖ Înmulțire



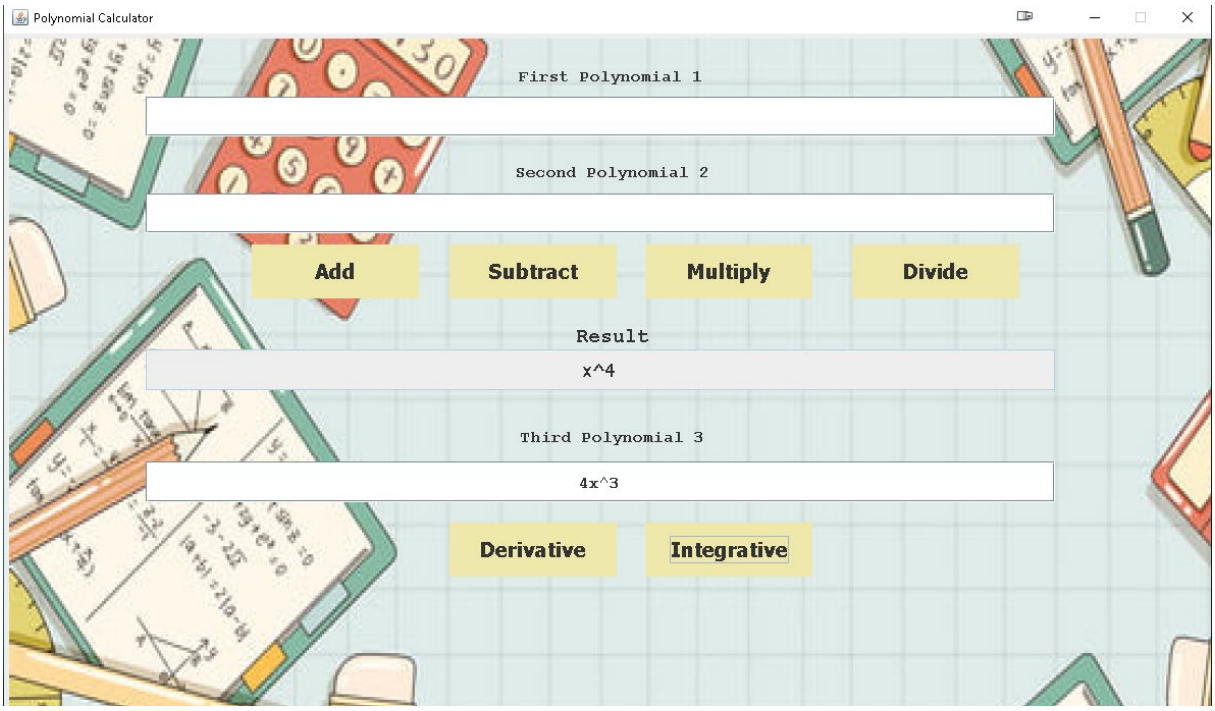
❖ Împărțire



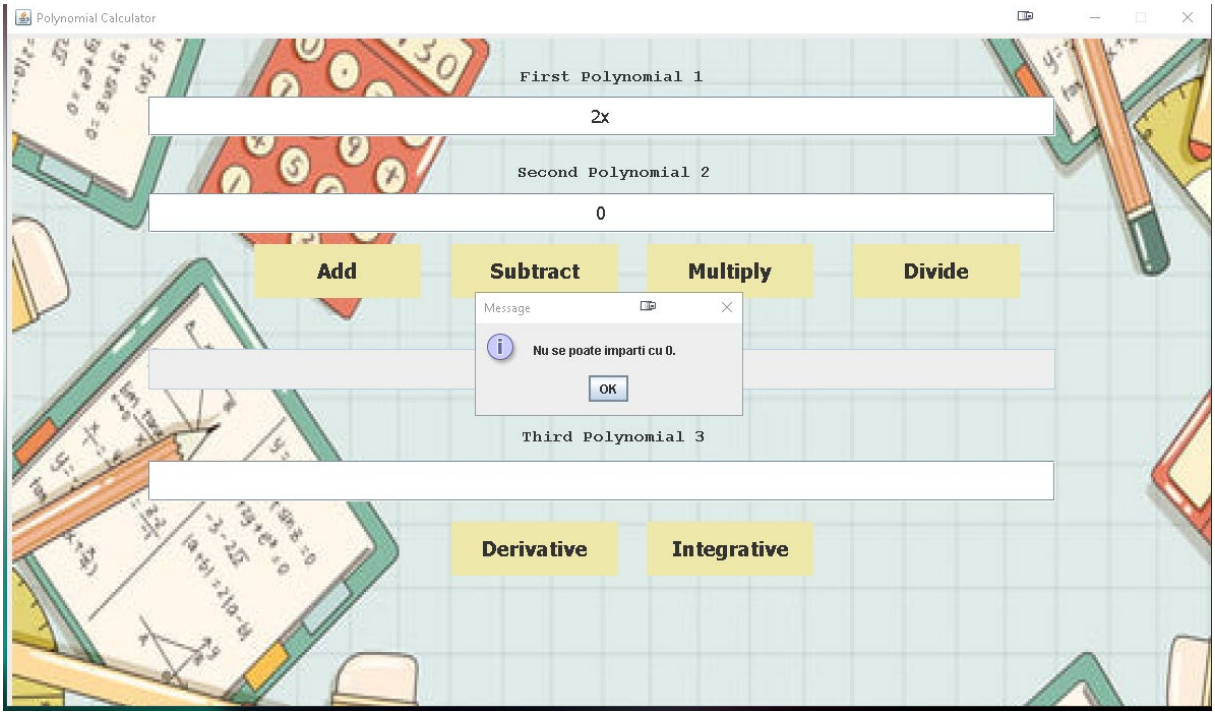
❖ Derivare



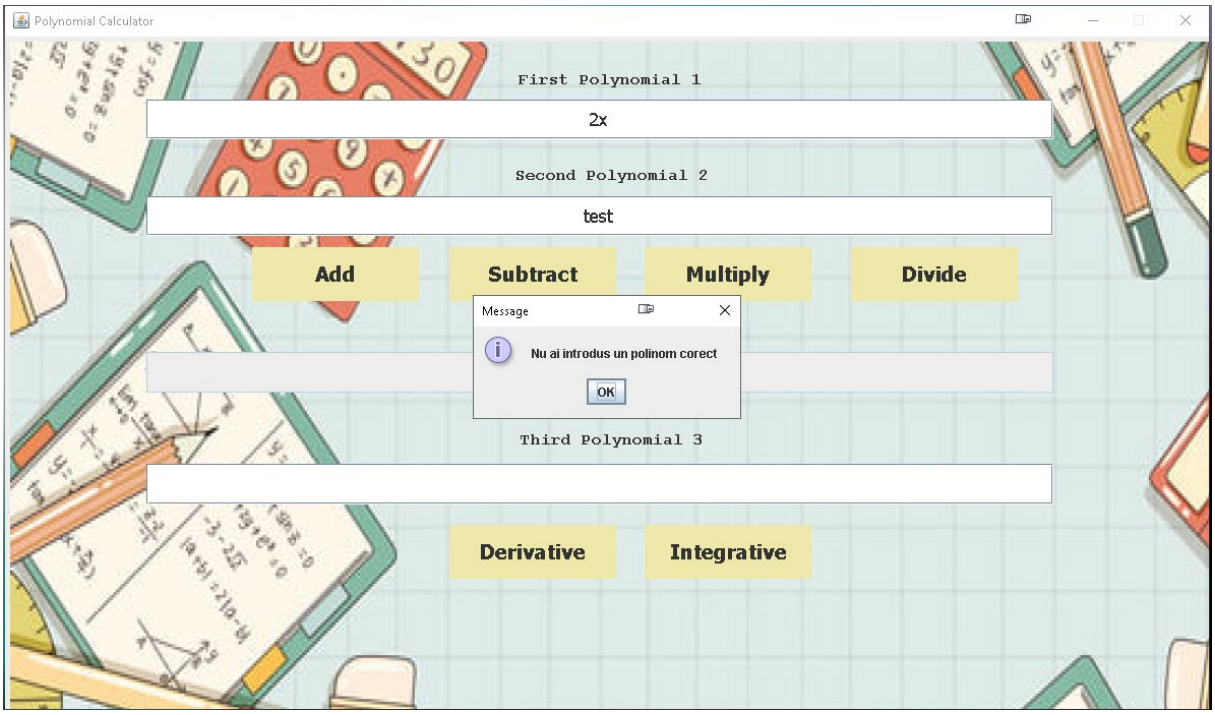
❖ Integrare



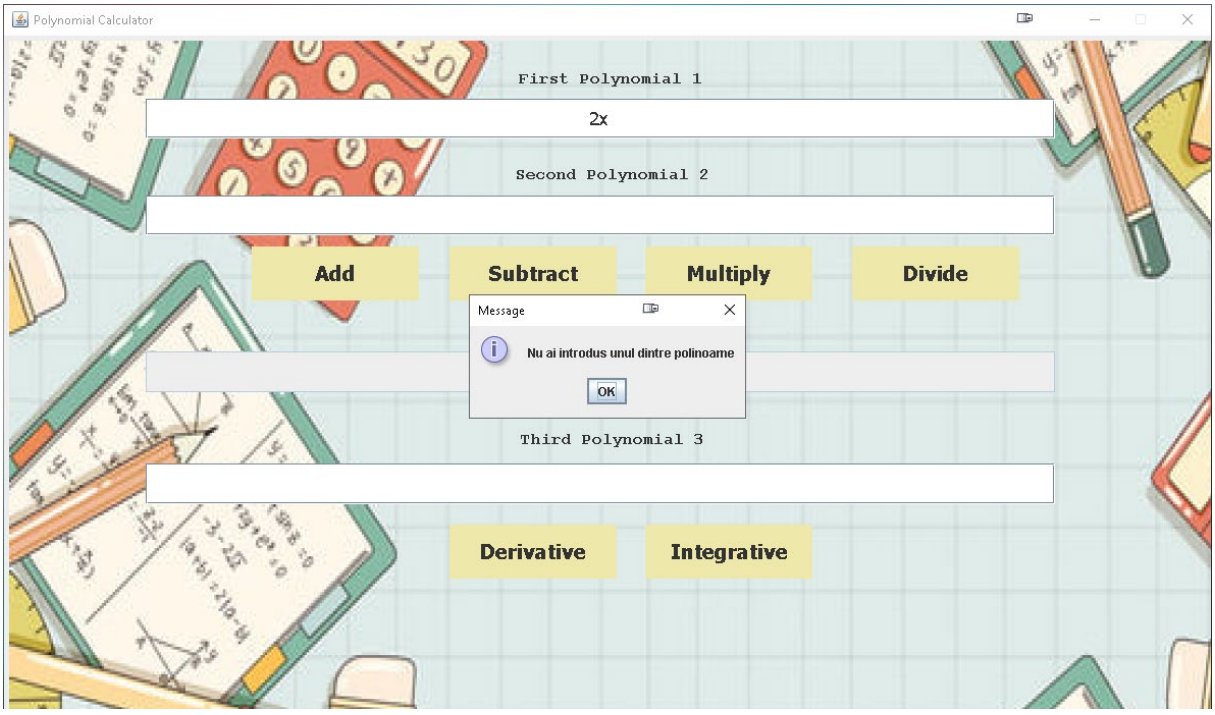
❖ Încercarea de împărțire prin 0



❖ **Introducerea unui polinom greșit**



❖ **Încercarea de a efectua o operație care necesită 2 polinoame cu un singur polinom**



6. Concluzii

În concluzie, acest proiect m-a ajutat să înțeleg mai bine MVC Pattern , să folosesc un Regex mai complex , implementarea paradigmelor POO și totodată m-a ajutat să fac debugging mai bine mai ales la algoritmii folosiți (ex. cel de împărțire) și să tratez cazuri excepționale.

Aplicația are o interfață user-friendly , chiar dacă nu este foarte complexă , dar efectuează toate operațiile de bază pe polinoame corect. Totuși , s-ar mai putea îmbunătății interfață grafică prin adăugarea unui meniu care să deschidaă câte un Panel separat pentru fiecare operație și eventual să fie făcută și pentru mobil sau pentru web.

7. Bibliografie

- **StackOverflow**
- **Regex101**
- **TutorialPoint – MVC Pattern**