

# Documentatie

## Tema 1 : Calculator de polinoame

Student: Radu Vlad

Grupa 30226

Materie: Tehnici de programare

Profesor indrumator: Mitrea Dan

# Cuprins

1. Obiectivul temei .
2. Analiza problemei, modelare, scenarii, cazuri de utilizare .
3. Proiectare .
4. Implementare .
  - a. Diagrame UML .
  - b. Clase si metode .
  - c. GUI .
5. Rezultate si Testare .
6. Concluzii .
7. Bibliografie .

# 1. Obiectivul temei .

Se dorește implementare și proiectarea unui calculator de polinoame. Calculatorul de polinoame poate realiza următoarele operații de bază : adunarea a două polinoame , scăderea a două polinoame , înmulțirea a două polinoame , împărțirea a două polinoame , cât și integrarea și derivarea pe structura polinoamelor. .

Fiecare dintre aceste polinoame vor fi introduse de către utilizator din interfața grafică , de unde acesta poate alege operația pe care dorește să o efectueze pentru cele două polinoame introduse anterior . De asemenea , rezultatul operației alese de utilizator va fi afișat în interfața grafică

## **Obiective secundare:**

- implementarea structurii polinomului ca listă înlanțuită de monoame
- implementarea monomului ca structură de date ce conține două numere, fiecare reprezentând coeficientul și exponentul
- implementarea unui regex care permite citirea polinomului de la tastatură și prin determinarea grupurilor construite generarea monoamelor ce urmează să fie adăugate în lista polinomului
- implementarea altor funcții ajutoare precum:
  - `toString()` = funcție ce permite afișarea polinomului într-un format prietenos cu utilizatorul.
  - `polinomSort(Polinom p)` = ordonarea polinomului în ordine descrescătoare în funcție de gradul exponentului.
  - `swap(Polinom p, Monom m, Monom n)` = interschimbă cele două monoame din polinomul p
  - `simplificaPolinom(Polinom p)` = aduce polinomul la cea mai simplă formă astfel încât monoamele să fie distincte după exponent

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

Cazuri de uiltizare:

Un utilizator foloseste programul prin interfata grafica care se deschide la executia programului. Astfel, acestuia ii este permis sa introduca doua polinoame de la tastatura si sa selecteze operatia pe care doreste sa o efectueze. Rezultatul este afisat in aceeasi fereastra.

Un mod usor de vizualizat a operatilor realizate de calculator este reprezentat printr-o diagrama Use Case, care prezinta succint modul in care utilizatorul intra in contact cu aplicatia: Acesta are access la partea de interfata grafica de unde poate efectua urmatoarele operatii :

Unde :

ADD – reprezinta operatia de adunare a doua polinoame

SUB – reprezinta operatia de scadere a doua polinoame

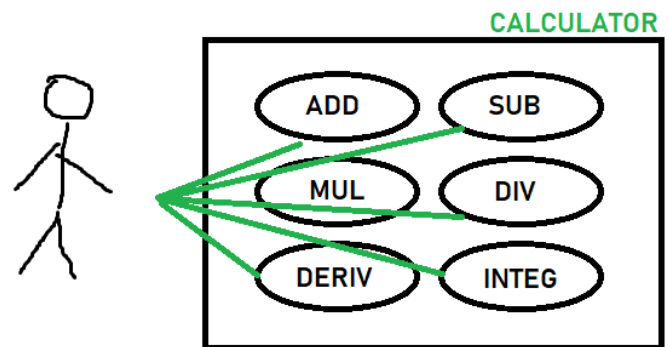
SUB – reprezinta operatia de scadere a doua polinoame

MUL – reprezinta operatia de inmultire a doua polinoame

DIV – reprezinta operatia de impartire a doua polinoame

DERIV – operatia de derivare a unui polinom

INTEG – operatia de integrare a unui polinom



### 3. Proiectare .

Polinoamele sunt construite din termeni numiti monoame, care sunt alcatuite dintr-o constanta numita coefficient inmultita cu una sau mai multe variabile. Fiecare variabila poate avea un exponent constant intreg pozitiv.

Exponentul unei variabile dintr-un monom este egal cu gradul acelei variabile in acel monom. Un monom fara variabile se numeste monom constant. Sau doar constanta. Coeficeintul unui monom poate fii orice numar, inclusiv fractii, numere irrationale sau negative.

Pentru proiectul current , am considerat un poliniom ce continue monoame avand coefficientul numere reale si o singura variabila in in constructia lor.

Pentru determinarea seturilor de monoame , am construit o structura numita regex care grupeaza elemenetele de tipul  $ax^b$  in grupuri din care extrag ulterior coefficientul a si exponentul b .

```
public Polinom(String sir){
    Pattern pattern = Pattern.compile("[+-]?[^\-+]+");
    Matcher matcher = pattern.matcher(sir);
    while (matcher.find()) {
        Monom m = new Monom(getCoefficient(matcher.group(1)), getExponent(matcher.group(1)));
        polinomul.add(m);
    }
}

private static int getCoefficient(String sir){
    String[] aux = sir.split(regex: "x");
    int coef = Integer.parseInt(aux[0]);
    return coef;
}

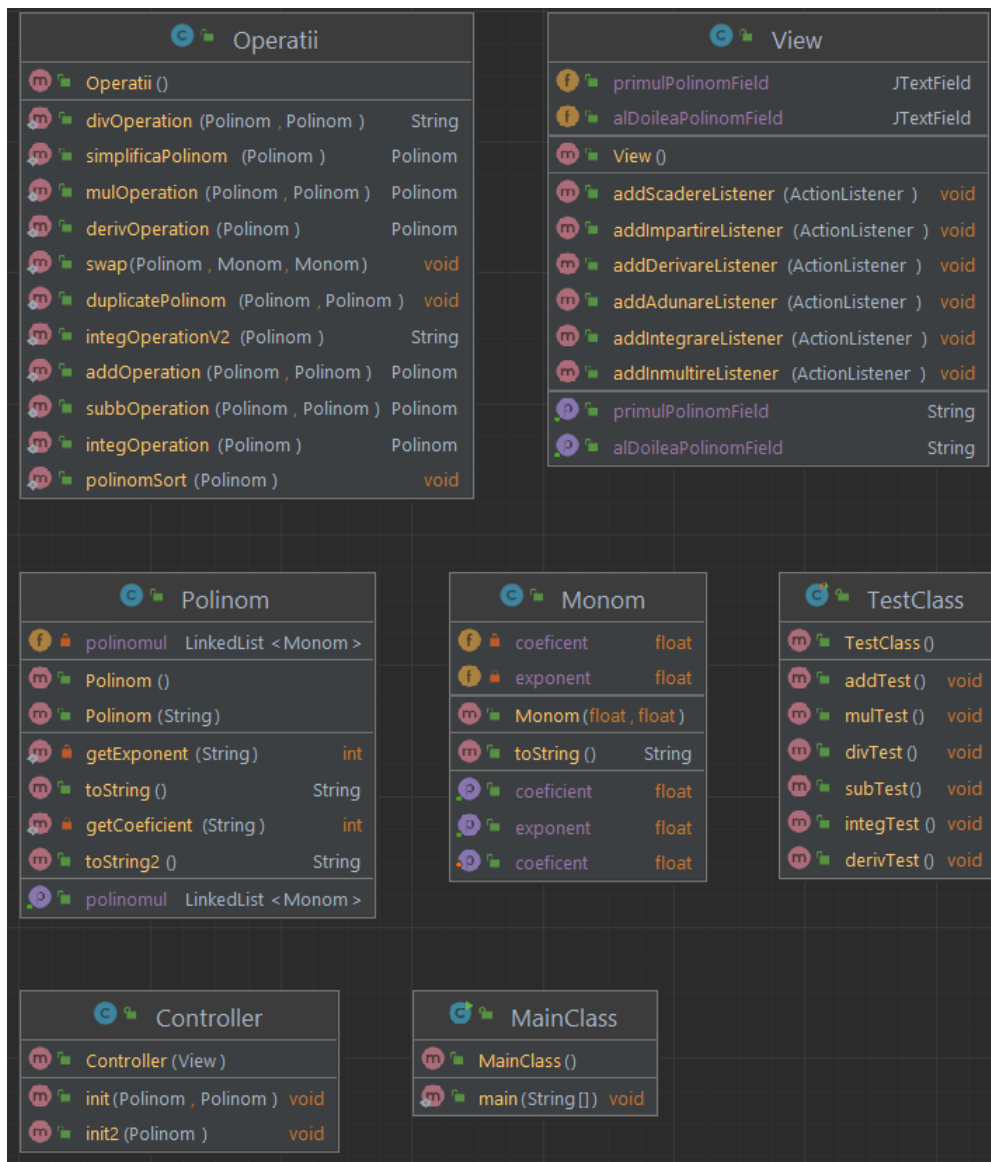
private static int getExponent(String sir){
    String[] aux = sir.split(regex: "\\^");
    int exponent = Integer.parseInt(aux[1]);
    return exponent;
}
```

Pentru ca programul sa ruleze corect, exista un set de reguli pe care utilizatorul trebuie sa il indeplineasca atunci cand introduce un polinom:

- Am ales ca necunoscuta tuturor polinoamelor sa fie 'x'
- Fiecare termen ( monom ) va fi de tipul :  $ax^b$  unde a este coefficientul si b exponentul monomului
- Termenii vor fi delimitati de semnele '+' sau '-' ;
- Nu se va pune spatiu intre termenii polinomului , sau intre semnele '+' sau '-' .
- Pentru primul monom din polinom,daca coefficientul este  $> 0$ , nu este nevoie sa se puna '+',dar nici nu e gresit daca se pune

## 4. Implementare .

### a) Diagrama UML .



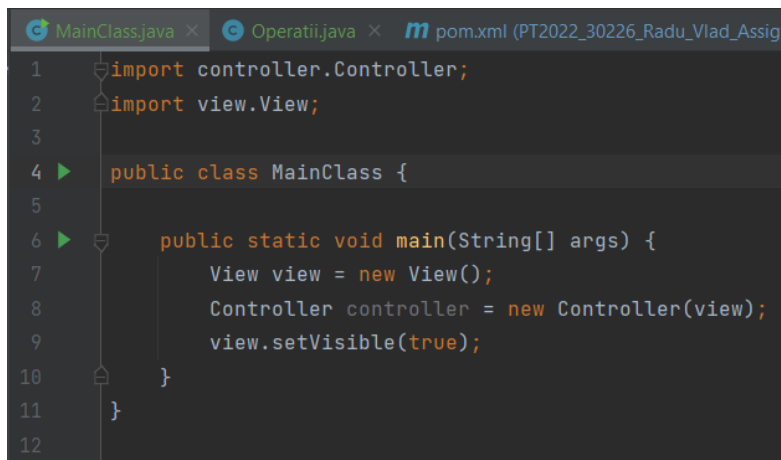
Unified Modeling Language (UML) joaca un rol important in dezvoltarea de software, dar si pentru sistemele non-software din numeroase domenii, deoarece este o modalitate prin care putem vizualiza comportamentul si structura unui sistem sau proces. UML ajuta la prezentarea posibilelor erori din structurile aplicatiilor, comportamentele sistemelor si alte procese.

Diagrama clasei este elementul principal al modelarii orientate pe obiecte.

## b) Clase si metode .

Pentru structurarea codului si pentru a respecta paradigmele din limbajul Java, am utilizat modelul MVC (Model, View, Controller), adica am impartit codul in 3 pachete la care am adaugat clasa Main si ulterior o clasa TestClass in care realizez testarea tuturor operatorilor efectuate.

Clasa MainClass contine o instanta a clasei View care controleaza partea de interfata grafica si o instanta a clasei Controller ce genereaza relatile dintre View si Model. Tot in clasa MainClass este setat vizibil calculatorul, Astfel la rularea programului utilizatorul are access la functionalitatile calculatorului.



```
1 import controller.Controller;
2 import view.View;
3
4 public class MainClass {
5
6     public static void main(String[] args) {
7         View view = new View();
8         Controller controller = new Controller(view);
9         view.setVisible(true);
10    }
11 }
12
```

Pachetul model contine cele 3 clase : clasa Polinom, clasa Monom si o clasa Operatii

Clasa Monom .

- are 2 atribute de tip real: coeficient si exponent, au vizibilitate de tip private si sunt instantiate in constructor.
- totodata pentru a avea access la atributele de tip private se folosesc niste metode getters si setters si o metoda toString() care afiseaza in mod natural monomul



```
package model;

public class Monom {

    private float coeficient;
    private float exponent;

    public Monom(float coeficient, float exponent){
        this.coeficient = coeficient;
        this.exponent = exponent;
    }

    public float getCoeficient() { return coeficient; }

    public float getExponent() { return exponent; }

    public void setCoeficient(float coeficient) { this.coeficient = coeficient; }

    public String toString() { return "coefficient: " + coeficient + " exponent: " + exponent; }
}
```

Clasa Polinom are un singur atribut de tipul private, o lista dublu inlantuita (Linked List <Monom>) care initial este initializata la null. Aceasta lista este accesata prin intermediul unui getter getPolinomul() , deasemenea am construit doua metode toString() care imi permit afisarea Polinomului intr-un mod natural . Cea de a doua afisare este folosita pentru operatia de Integrare descrisa ulterior.

```
public LinkedList<Monom> getPolinomul() { return this.polinomul; }

public String toString() {
    String rezultat = "";
    for (Monom m : polinomul) {
        if(rezultat.length() == 0) {
            if(m.getCoefficient() > 0)
                rezultat += (int) m.getCoefficient()+"x^"+ (int) m.getExponent();
            else
                if(m.getCoefficient() < 0)
                    rezultat += (int) m.getCoefficient()+"x^"+ (int) m.getExponent();
        }
        else
            if(m.getCoefficient() > 0)
                rezultat += "+"+(int) m.getCoefficient()+"x^"+ (int) m.getExponent();
            else
                if(m.getCoefficient() < 0)
                    rezultat += (int) m.getCoefficient()+"x^"+ (int) m.getExponent();
    }
    return rezultat;
}

public String toString2() {...}
```

Tot in clasa Polinom , am adaugat un constructor Polinom( String sir) care primeste ca parametru polinomul reprezentat sub forma unui sir iar prin intermediul unui regex sirul se imparte in grupuri. Aceste grupuri la randul lor sunt impartite in coeficient si exponent . Se formeaza un monom pe care il adaug in lista polinomului.

```
public Polinom(String sir){
    Pattern pattern = Pattern.compile("[+-]?[^-+]+");
    Matcher matcher = pattern.matcher(sir);
    while (matcher.find()) {
        Monom m = new Monom(getCoefficient(matcher.group(1)), getExponent(matcher.group(1)));
        polinomul.add(m);
    }
}

private static int getCoefficient(String sir){
    String[] aux = sir.split(regex: "x");
    int coef = Integer.parseInt(aux[0]);
    return coef;
}

private static int getExponent(String sir){
    String[] aux = sir.split(regex: "\\^");
    int exponent = Integer.parseInt(aux[1]);
    return exponent;
}
```



Clasa Operatii , este clasa ce imi contine toate operatiile pe doresc sa le fac pe Polinoame. Toate metodele sunt statice, acestea se apeleaza prin intermediul clasei .

ADUNAREA :

```
public static Polinom addOperation(Polinom p, Polinom q) {
    Polinom rezultat = new Polinom();

    for(Monom m : p.getPolinomul()) {
        int ok = 0;
        for(Monom n : q.getPolinomul()) {
            if(m.getExponent() == n.getExponent()) {
                ok = 1;
                if(m.getCoefficient() + n.getCoefficient() != 0) {
                    Monom x = new Monom( coefficient: m.getCoefficient()+n.getCoefficient(), m.getExponent());
                    rezultat.getPolinomul().add(x); }
            }
        }
        if(ok == 0)
            rezultat.getPolinomul().add(m);
    }

    for(Monom n : q.getPolinomul()) {
        int ok = 0;
        for(Monom m : p.getPolinomul())
            if(n.getExponent() == m.getExponent())
                ok = 1;
        if(ok == 0)
            rezultat.getPolinomul().add(n);
    }

    polinomSort(rezultat);
    return rezultat;
}
```

Algoritmul de adunare functioneaza astfel:

Parcurgem cu ajutorul a doua for-each uri cele doua Polinoame p, q iar daca monomul m din p are exponentul egal cu monomul n din q in Polinomul rezultat se adauga un nou monom x reprezentand suma celor doua monoame m si n, in caz contrar inseamna ca monomul m nu se regaseste in q si il adaugam in rezultat. Parcurgem inca o data polinomul q ca sa ne asiguram ca adaugam si termenii care nu se regasesc in p. La final sortam polinomul rezultat descator dupa exponent.

Exemplu daca avem polinomul  $p = 2x^2 + 3x + 7$  si polinomul  $q = 3x^3 + x^2$

Initial polinomul rezultat va contine doar  $rez = 3x^2 + 4x + 7$  urmand ca dupa ultima parcurgere sa se adauge  $3x^3$  la final si prin intermediul lui polinomSort( rez) este sortat descrescator rezultatul.

## SCADEREA :

```
public static Polinom subbOperation (Polinom p, Polinom q) {
    Polinom rezultat = new Polinom();

    for(Monom m : p.getPolinomul()) {
        int ok = 0;
        for(Monom n : q.getPolinomul()) {
            if(m.getExponent() == n.getExponent()) {
                ok = 1;
                if(m.getCoefficient() - n.getCoefficient() != 0) {
                    Monom x = new Monom( coefficient: m.getCoefficient() - n.getCoefficient(), m.getExponent());
                    rezultat.getPolinomul().add(x);
                }
            }
        }
        if(ok == 0)
            rezultat.getPolinomul().add(m);
    }

    for(Monom n : q.getPolinomul()) {
        int ok = 0;
        for(Monom m : p.getPolinomul())
            if(n.getExponent() == m.getExponent())
                ok = 1;
        if(ok == 0){
            Monom x = new Monom( -n.getCoefficient(), n.getExponent());
            rezultat.getPolinomul().add(x);
        }
    }

    polinomSort(rezultat);
    return rezultat;
}
```

Algoritmul de scadere are o analogie identica cu cel de adunare. La fel, se parcurg cele doua Polinoame p, q iar termenii cu acelasi grad al exponentului obtinuti din monoame sunt calculati si adaugati impreuna in intr-un monom nou pe care in inseram in polinomul rezultat . Monoamele ce nu sunt comune sunt adaugate la sfarsitul Polinomului in ordinea descoperirii. In final Polinomul este sortat descrescator folosind tot functia polinomSort( rezultat);

## INMULTIREA :

```
public static Polinom mulOperation(Polinom p, Polinom q) {
    Polinom rez = new Polinom();

    for(Monom m : p.getPolinomul()) {
        for(Monom n : q.getPolinomul()) {
            Monom k = new Monom( coefficient: m.getCoefficient()*n.getCoefficient(), exponent: m.getExponent()+n.getExponent());
            rez.getPolinomul().add(k);
        }
    }
    polinomSort(rez);
    return simplificaPolinom(rez);
}
```

Se construiesc un nou Polinom rez reprezentand rezultatul inmultirii initializat la null. Parcurgem integral cele doua Polinoame p, q adaugand in rez un nou Monom k construit prin inmultirea coeficientilor si insumararea exponentilor celor doua monoame din p si q.

In final se sorteaza descrescator polinomul, dupa gradul exponentului si se simplifica prin restrangerea monoamelor care au gradul egal. Polinomul rezultat are monoame cu grade distincte.

De exemplu daca avem polinomul  $p = 2x^3 + 1x^1$  si polinomul  $q = 1x^2 + 2x^1 + 3x^0$

Rezultatul intermediar este:  $2x^5 + 4x^4 + 6x^3 + 1x^3 + 2x^2 + 3x^1$  care se simplifica in:  $2x^5 + 4x^4 + 7x^3 + 2x^2 + 3x^1$

## IMPARTIREA :

```
public static String divOperation(Polinom p, Polinom q){
    Polinom catul = new Polinom();
    Polinom p1 = new Polinom();
    duplicatePolinom(p1, p);
    Polinom intermediar = new Polinom();

    while(p1.getPolinomul().get(0).getExponent() >= q.getPolinomul().get(0).getExponent()){
        float a = p1.getPolinomul().get(0).getCoefficient() / q.getPolinomul().get(0).getCoefficient();
        float b = p1.getPolinomul().get(0).getExponent() - q.getPolinomul().get(0).getExponent();
        Monom m = new Monom(a, b);
        catul.getPolinomul().add(m);

        intermediar.getPolinomul().clear();
        intermediar.getPolinomul().add(m);

        Polinom x = new Polinom();
        Polinom y = new Polinom();
        duplicatePolinom(x, mulOperation(q, intermediar));
        duplicatePolinom(y, subbOperation(p1, mulOperation(q, intermediar)));

        p1 = new Polinom();
        duplicatePolinom(p1, y);
    }

    return "catul: " + catul.toString() + "\nrestul: " + p1.toString();
}
```

Pentru impartire am folosit algoritmul prezentat in indrumator. Astfel, Cat timp gradul polinomului p este mai mare sau egal decat gradul polinomului q se impart monoamele de pe prima pozitie adica pozitia 0. Acest monom rezultat se adauga in Polinomul numit cat reprezentand primul termen al catului. Totodata valoarea polinomului p este inlocuita cu vechea valoare minus produsul dintre noul monom adaugat in cat si deimpartitul adica polinomul q.

Am folosit o copie a polinomului p astfel incat valoarea acestuia sa ramana nemodificata. Functia duplicatePolinom(Polinom p, Polinom q) adauga in p valoarea din q.

## DERIVAREA :

```
public static Polinom derivOperation(Polinom p) {
    Polinom rez = new Polinom();

    for(Monom m : p.getPolinomul()){
        Monom n = new Monom( coefficient: m.getCoefficient() * m.getExponent(), exponent: m.getExponent() - 1);
        if(n.getExponent() >= 0)
            rez.getPolinomul().add(n);
    }

    return rez;
}
```

Se construiesc un nou polinom rez. Derivarea parcurge fiecare monom m din polinomul p linear, contine un nou monom n avand coeficientul egal cu produsul dintre coeficientul si exponentul polinomului m si exponentul scazut cu o unitate, pe acest polinom il adauga in polinomul rezultat.

Daca avem polinomul  $p = 4x^2 + 3x^1 + 6x^0$  rezultatul va fi egal cu  $rez = 8x^1 + 3x^0$

Monoamele ca initial aveam exponentul 0 adica erau formate dintr-o constanta nu vor fi adaugate in polinomul derivat

## INTEGRAREA :

```
public static Polinom integOperation(Polinom p) {
    Polinom rez = new Polinom();

    for(Monom m : p.getPolinomul()) {
        Monom n = new Monom( coefficient: m.getCoeficient() * 1 / (m.getExponent()+1), exponent: m.getExponent()+1);
        rez.getPolinomul().add(n);
    }
    return rez;
}
```

Pentru integrare am implementat doua metode, o metoda care foloseste coeficienti de tip float si realizea impartirea dintre coeficientul si exponentul monomului. Astfel dupa efectuarea integrarii se poate testa derivarea pe polinomul rezultat.

```
public static String integOperationV2(Polinom p) {
    String rez = "";
    for(Monom m : p.getPolinomul())
        if(m.getCoeficient() > 0)
            rez += "+" + (int)m.getCoeficient() + "/" + (int)(m.getExponent()+1) + "x^" + (int)(m.getExponent()+1);
        else
            rez += (int)m.getCoeficient() + "/" + (int)(m.getExponent()+1) + "x^" + (int)(m.getExponent()+1);
    return rez;
}
```

Cea de a doua metoda afiseaza intr-un mod natural sub forma de fractie rezultatul integrarii polinomului p, parcurgand lista de monoame si punand intre coeficient si (exponent + 1) "/" reprezentand o fractie.

## ALTE METODE :

```
public static void polinomSort(Polinom p){
    for(int i = 0; i < p.getPolinomul().size(); i++)
        for(int j = i+1; j < p.getPolinomul().size(); j++)
            if(p.getPolinomul().get(i).getExponent() < p.getPolinomul().get(j).getExponent())
                swap(p, p.getPolinomul().get(i), p.getPolinomul().get(j));
}

public static void swap(Polinom p, Monom m, Monom n){
    int index1 = p.getPolinomul().indexOf(m);
    int index2 = p.getPolinomul().indexOf(n);
    p.getPolinomul().set(index1, n);
    p.getPolinomul().set(index2, m);
}
```

polinomSort ( Polinom p ) = se bazeaza pe algoritmul bubble sort, acesta parcurge toata lista de monoame a polinomului si interchimba valorile care au exponent mai mare decat monomul current, folosindu-se de functia swap.

Swap( Polinom p, Monom m, Monom n ) = interchimba cele doua monoame in lista polinomului p pe baza indexului la care se afla fiecare dintre cele doua monoame.

```

public static Polinom simplificaPolinom(Polinom p){
    Polinom rezultat = new Polinom();
    int i=0;

    while(i < p.getPolinomul().size()-1){
        Monom m = new Monom(p.getPolinomul().get(i).getCoeficient(), p.getPolinomul().get(i).getExponent());
        i++;
        while(m.getExponent() == p.getPolinomul().get(i).getExponent()){
            m.setCoeficient(m.getCoeficient() + p.getPolinomul().get(i).getCoeficient());
            i++;
        }
        rezultat.getPolinomul().add(m);
    }
    rezultat.getPolinomul().add(p.getPolinomul().get(i));
    return rezultat;
}

```

Pentru a folosi functia simplificaPolinom ( Polinom p ) se apeleaza initial polinomSort(Polinom p) pentru a sorta polinomul. Se presupune ca polinomul este sortat, pentru fiecare Monom m se percurg toti vecinii care au exponentul egal si se adauga valoarea coeficientilor in monomul m, in Polinomul rezultat se adauga valoarea finala a monomului m.

```

public static void duplicatePolinom(Polinom p, Polinom q){
    for(Monom m : q.getPolinomul())
        p.getPolinomul().add(m);
}

```

duplicatePolinom( Polinom p, Polinom q ) parcurge lista polinomului q pe care o adauga in polinomul p.

## c) GUI

Partea de gui este impartita in cele doua pachete

view: care implementeaza partea grafica a interfeței cu toate componentele ferestrei deschise la executia programului, dar si declararea Action Listenerilor care fac legatura dintre view si controller.

controller: contine clase necesare functionarii butoanelor, aceasta prelucreaza datele din model si le trimite inapoi spre view.

\*declararea text fieldurilor a butoanelor si a metodelor ActionListener din view\*

```
public class View extends JFrame {  
  
    private JLabel primulPolinomLabel;  
    public JTextField primulPolinomField;  
    private JLabel alDoileaPolinomLabel;  
    public JTextField alDoileaPolinomField;  
    private JLabel rezultatLabel;  
    public JTextPane rezultatText;  
  
    private JButton addButton;  
    private JButton subButton;  
    private JButton mulButton;  
    private JButton divButton;  
    private JButton derivateButton;  
    private JButton integrateButton;  
  
    public View() {  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        this.setSize( width: 400, height: 300);  
    }  
}
```

```
public String getPrimulPolinomField() { return primulPolinomField.getText(); }  
  
public String getAlDoileaPolinomField() { return alDoileaPolinomField.getText(); }  
  
public void addAdunareListener(ActionListener e1) { addButton.addActionListener(e1); }  
  
public void addScadereListener(ActionListener e2) {...}  
  
public void addInmultireListener(ActionListener e3) { mulButton.addActionListener(e3); }
```

Clasa Controller instanteaza toate attributele clasei View in constructor, astfel ca la apasarea unui buton de catre utilizator prin controller programul stie sa efectueze operatiile necesare.

```
public Controller(View view) {  
    this.view = view;  
    view.addAdunareListener(new AdunareListener());  
    view.addScadereListener(new ScadereListener());  
    view.addInmultireListener(new InmultireListener());  
    view.addImpartireListener(new ImpartireListener());  
    view.addDerivareListener(new DerivareListener());  
    view.addIntegrareListener(new IntegrareListener());  
}  
  
public void init(Polinom p, Polinom p){...}  
  
public void init2(Polinom p){...}  
  
class AdunareListener implements ActionListener {...}  
  
class ScadereListener implements ActionListener {...}  
  
class InmultireListener implements ActionListener {...}
```

Așa arată implementarea părții grafice construită dintr-o parte superioară ce marchează datele ce trebuie introduse la tastatură adică primul și cel de al doilea polinom.

Partea inferioară este reprezentată de cele 6 butoane care prin apăsarea lor determină efectuarea operației și afișarea rezultatului din textfield.

## 5 Rezultate și Testare

Partea de testare a fost realizată cu ajutorul bibliotecii JUnit. Am construit o clasă care să conțină un test pentru fiecare dintre operațiile efectuate: adunare, scădere, înmulțire, împărțire, derivare, integrare și folosind funcția `AssertEquals` se testează dacă rezultatul operației este același cu rezultatul la care ne așteptăm.

```
@Test
public void addTest() {
    Polinom p = new Polinom( sir: "4x^5-3x^4+1x^2-8x^1+1x^0");
    Polinom q = new Polinom( sir: "3x^4-1x^3+1x^2+2x^1-1x^0");
    String rez = new String( original: "4x^5-1x^3+2x^2-6x^1");
    assertEquals(rez, Operatii.addOperation(p, q).toString());
}
```

-exemplu de test pentru operația de Adunare

✓ Test Results	24 ms
✓ TestClass	24 ms
✓ subTest()	18 ms
✓ addTest()	
✓ integTest()	4 ms
✓ derivTest()	
✓ mulTest()	
✓ divTest()	2 ms

-în cazul în care operațiile funcționează corect astfel încât rezultatul să fie cel așteptat, testul o să fie marcat ca trecut având o bifă verde.

## 5. Concluzii

In concluzie, precizez ca proiectul realizat: calculator de polinoame are un domeniu vast de aplicare atat in domeniul stiintific: matematica, fizica cat si in in viata cotidiana usurand semnificativ calculele necesare obtinerii unui rezultat.

Totodata proiectul poate fi imbunatatit astfel incat sa citeasca date de la tastatura intr-un limbaj mult mai familiar cu mediul in care scrie utilizatorul obisnuit:

De la :  $(a_n)x^n + (a_{n-1})x^{(n-1)} + \dots + a_1 x^1 + a_0 x^0$  la un mod mai natural de scriere aplicat si pe cazurile speciale in care lipsete un termen.

Personal proiectul m-a ajutat sa imi amintesc tehnicile de programare invatate semestrul trecut, cat si sa imi dezvolt abilitatile de a folosi noi librarii si de programa aplicatii de complexitate mai ridicata.

## 6 Bibliografie

[https://dsrl.eu/courses/pt/materials/A1\\_Support\\_Presentation.pdf](https://dsrl.eu/courses/pt/materials/A1_Support_Presentation.pdf)  
<https://stackoverflow.com/>  
<https://www.geeksforgeeks.org/>