



# *Calculator de polinoame*

## **Documentatie**

**MOLDOVAN ADELINA-STEFANIA**

**GRUPA 30226**

## CUPRINS

1. Obiectivul temei . . . . .	3
1.1 Obiectivul principal . . . . .	3
1.2 Obiectivul secundar . . . . .	3
2. Analiza problemei, modelare, scenario, cazuri de utilizare . . . . .	4
2.1 Use case-uri si scenarii . . . . .	4
3. Proiectare . . . . .	5
3.1 Diagrame UML . . . . .	5
3.2 Structuri de date . . . . .	5
3.3 Interfata utilizator . . . . .	6
3.4 Packages . . . . .	6
4. Implementare. . . . .	7
4.1 Clasa Monom. . . . .	7
4.2 Clasa Polinom. . . . .	8
4.3 Clasa Model. . . . .	8
4.4 Clasa View. . . . .	10
4.5 Clasa Controller. . . . .	11
4.6 Clasa MVC. . . . .	11
5. Rezultate. . . . .	12
6. Concluzii. . . . .	12
7. Bibliografie. . . . .	13

# 1. Obiectivul temei

## Obiective principale

Proiectarea și implementarea unui calculator polinomial cu o interfață grafică dedicată prin intermediul căreia utilizatorul poate introduce polinoame, selecta operația care trebuie efectuată (adică adunarea, scăderea, înmulțire, împartire, derivare, integrare) și afișați rezultatul .

## Obiective secundare

- 1 . - dezvoltarea de use case-uri : folosim use case-uri (caz de utilizare) pentru a reprezenta cerințe ale utilizatorilor. Sunt descrise acțiuni pe care un program le execută atunci când interacționează cu actori și care conduc la obținerea unui rezultat .
- 2 . - dezvoltarea de diagrame UML : folosim diagrame UML pentru a înregistra relațiile dintre clase .
- 3 . - implementarea unor clase în java : sunt descrise clasele folosite și rolul acestora .
- 4 . – dezvoltarea algoritmilor : procesul de dezvoltare al algoritmilor utilizați pentru acest proiect .
- 5 . – șablonul MVC ( Model – View – Controller ) : Separarea programului în model, view ( creează afișajul ) și controller ( răspunde la cererile utilizatorului ) .

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

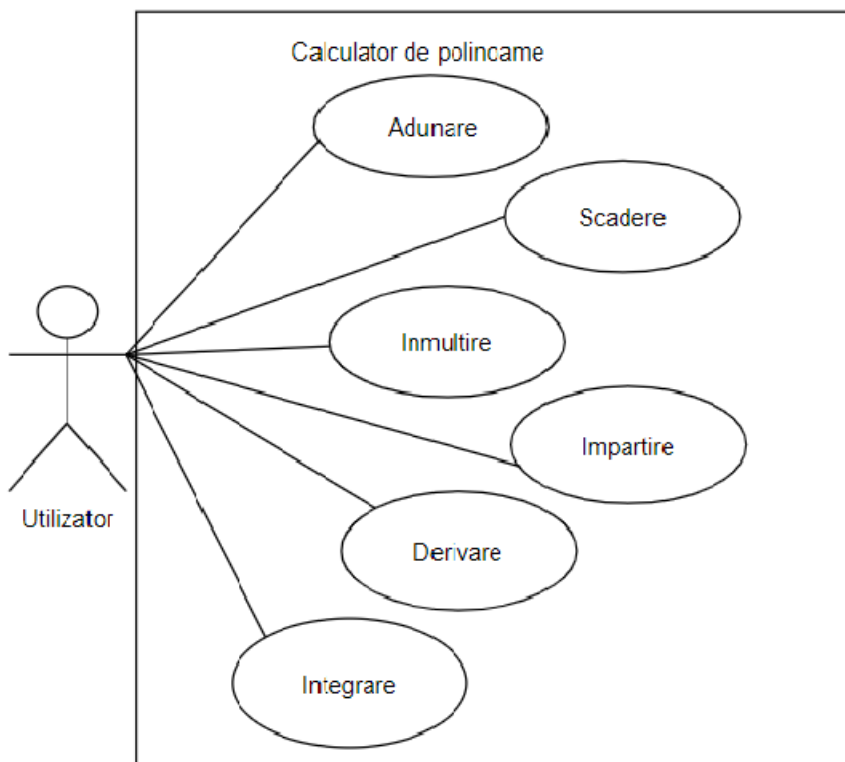
### Use case-uri

O diagramă use case este una din diagramele folosite în UML pentru a modela aspectele dinamice ale unui program alături de diagrama de activități, diagrama de stări, diagrama de secvență și diagrama de colaborare .

Elementele componente ale unei diagrame use case sunt:

- use case-uri ;
- actori ;
- relațiile care se stabilesc între use case-uri, între actori și între use case-uri și actori .

Un use case (caz de utilizare) reprezintă cerințe ale utilizatorilor. Este o descriere a unei mulțimi de secvențe de acțiuni (incluzând variante) pe care un program le execută atunci când interacționează cu entitățile din afara lui (actori) și care conduc la obținerea unui rezultat observabil și de folos actorului .



Exemplu descriere use case :

Use case : adunare

Actor: utilizatorul

Scenariu:

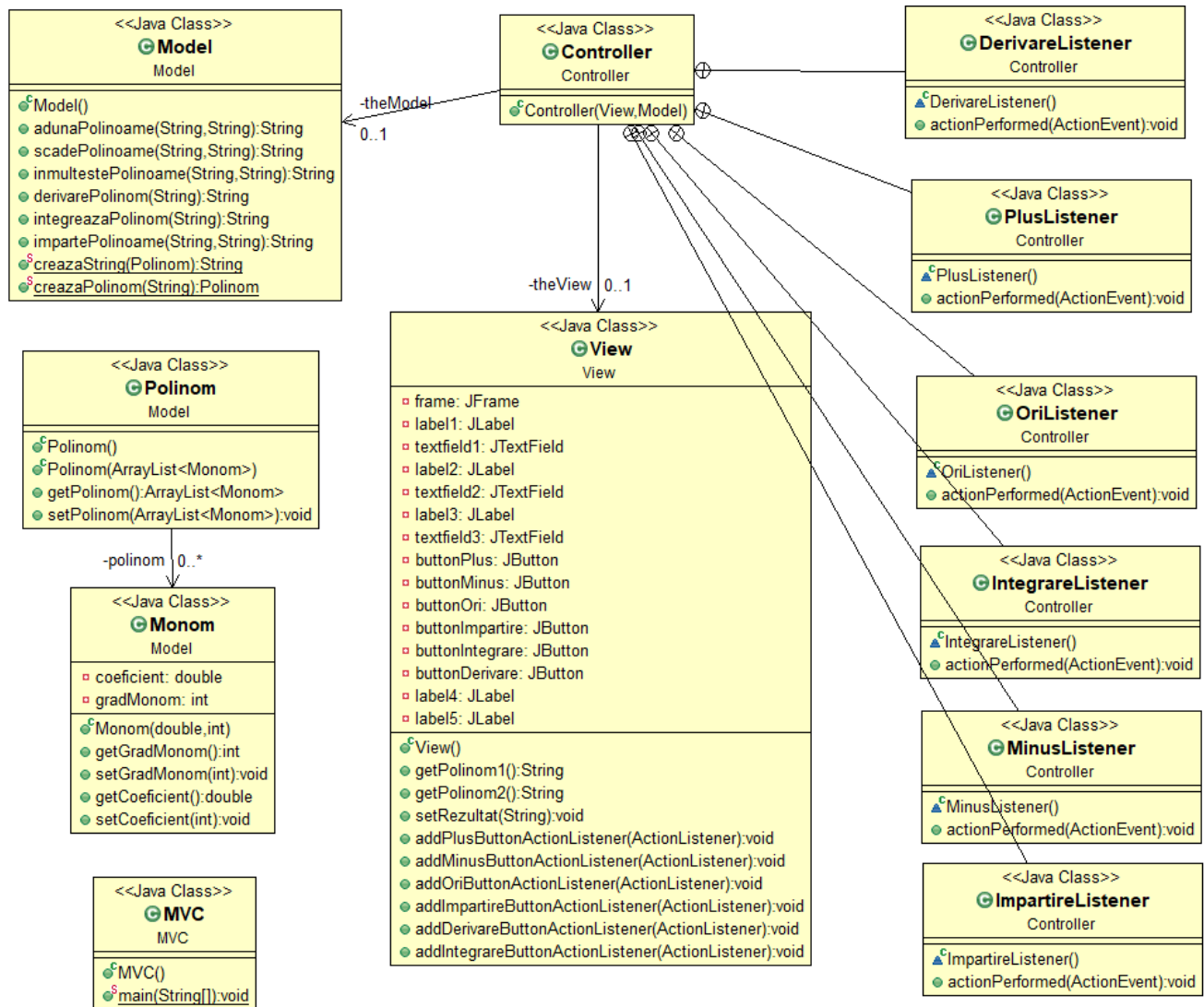
1. Utilizatorul introduce primul polinom
2. Utilizatorul introduce al doilea polinom
3. Utilizatorul alege operatia dorita , in acest caz, adunare
4. Utilizatorul poate vizualiza rezultatul

Alte variante:

- Utilizatorul poate alege alta operatie; ex : scadere, inmultire, impartire, derivare, integrare

### 3.Proiectare

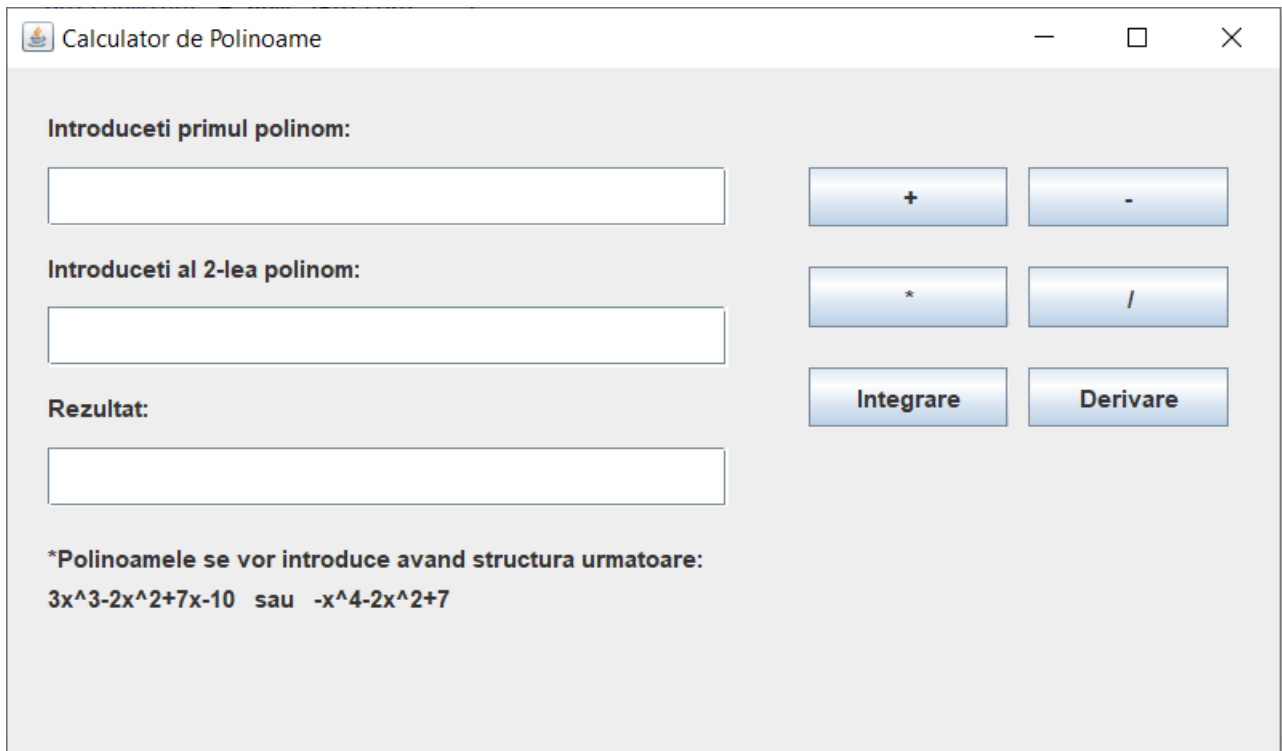
#### Diagrama UML



#### Structuri de date

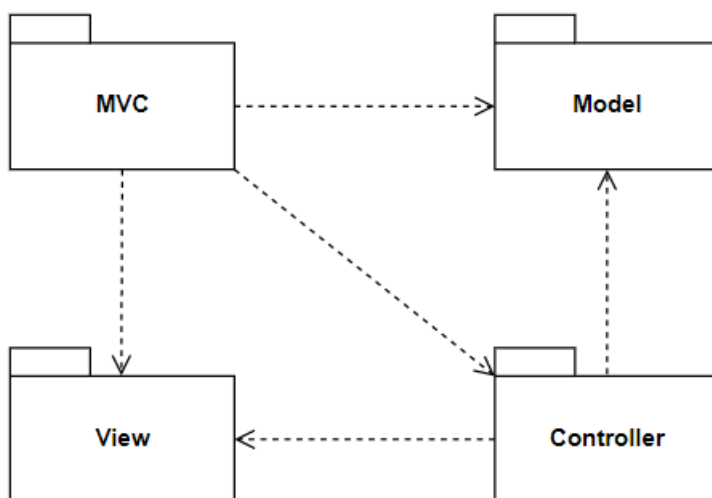
Structurile de date folosite in acest proiect sunt ArrayList -urile, care contin elemente de tipul Monom pentru a memora un polinom si pentru a putea efectua mai usor si mai eficient operatiile specifice. Deci, ArrayList-urile vor fi folosite pentru toate operatiile din cadrul programului, acestea memorand elementele care trebuie adunate, scazute , inmultite , impartite , derivate sau integrate . ArrayList-ul este folosit si pentru a retine rezultatul calculului efectuat in urma operatiei alese, dupa care va fi prelucrat intr-un string si afisat.

## Interfata utilizator



Prin intermediul interfetei se realizeaza comunicarea cu utilizatorul. Dintre pachetele care contin elemente de grafica am folosit `javax.swing`, care vine cu o gama extinsa de componente si facilitati .

## Packages



Pachetul (package) este o grupare de elemente ale unui model și reprezintă baza necesară controlului configurației, depozitării și accesului . Este un container logic pentru elemente între care se stabilesc legături . Toate elementele UML pot fi grupate în pachete (cel mai des pachetele sunt folosite pentru a grupa clase). Un element poate fi conținut într-un singur pachet

## 4.Implementare

Calculatorul este organizat conform șablonului Model-View-Controller (MVC) . Ideea este separarea programelor în Model, View (vedere, vizualizare) (crează afișajul, interacționând cu Modelul după nevoi), și Controller (răspunde la cererile utilizatorului, interacționând atât cu Vizualizarea cât și cu Modelul) .

Rolul elementelor din arhitectura MVC :

- Model – încapsulează datele specifice unei aplicații și definește logica și computațiile care manipulează și procesează datele respective . Modelul nu știe despre vederi și controloare. Când se schimbă, în mod tipic modelul notifică observatorii săi despre schimbare.
- Vederea – redă conținutul unui model. Specifică exact cum se prezintă utilizatorului datele din model. Dacă modelul se schimbă, vederea trebuie să-și actualizeze prezentarea după nevoi.
- Controlorul – traduce interacțiunile utilizatorului cu vederea în acțiuni pe care le va executa modelul. Într-un client GUI de sine stătător, interacțiunile pot fi click-uri pe butoane, selecții de meniu, introducerea unor date etc.

Pachetul Model contine clasele : Monom, Polinom si Model . Clasele Monom si Polinom definesc structurile necesare pentru calcule .

### Clasa Monom

- contine doua attribute care descriu un monom : coeficient de tipul double si gradMonom de tipul int;
- contine gettere si settere , metode care permit citirea si scrierea coeficientului si gradului;
- aceasta clasa tine structura principala cu care se va lucra, polinoamele fiind alcatuite din mai multe monoame;

```
public class Monom {  
  
    private double coeficient;  
    private int gradMonom;  
  
    public Monom(double coeficient, int gradMonom) {  
        this.coeficient = coeficient;  
        this.gradMonom = gradMonom;  
    }  
  
    public int getGradMonom() {  
        return gradMonom;  
    }  
  
    public void setGradMonom(int gradMonom) {  
        this.gradMonom = gradMonom;  
    } ...  
}
```

### Clasa Polinom

- contine un singur atribut: polinom, care este un ArrayList de monoame;
- contine metode care permit scrierea si citirea arrayList-ului de monoame;

```
public class Polinom {  
  
    private ArrayList<Monom> polinom;  
  
    public Polinom(){  
  
    }  
  
    public Polinom(ArrayList<Monom> polinom) {  
        this.setPolinom(polinom);  
    } ...  
}
```

### Clasa Model

- este clasa care se ocupa de operatiile pe polinoame, asadar contine metode pentru fiecare din cele sase operatii;
- de asemenea contine alte 2 functii care se ocupa de prelucrarea datelor de intrare si iesire; o metoda creeazaPolinom, care are ca parametru un String si returneaza un obiect de tipul Polinom, metoda care converteste stringul introdus intr-un obiect din clasa Polinom(arrayList de monoame); o alta metoda care prelucreaza date este creeazaString care converteste un obiect de tipul Polinom intr-un string, pentru a putea afisa rezultatul;

Exemplu pentru metoda adunaPolinoame:

- functia primeste ca argumente doua obiecte de tipul String si returneaza tot un String
- stringurile primite ca argumente vor fi transformate in obiecte de tipul Polinom cu ajutorul metodei creeazaPolinom
- se vor parcurge ambele polinoame in acelasi timp, si daca exista doua puteri egale se aduna coeficientul, in caz contrar se incrementeaza indexul pentru unul dintre cele doua polinoame, in functie de caz
- rezultatul adunarii este memorat intr-un arrayList de monoame
- la final se creeaza un obiect de tipul polinom, care va fi instantiat cu arrayList-ul de monoame
- cu ajutorul functiei creeazaString , polinomul creat la final va fi transformat intr-un string, pentru a putea fi afisat
- mai jos avem un exemplu de cod pentru functia adunaPolinoame



Exemplu de cod pentru functia adunaPolinoamej :

```

public String adunaPolinoame(String pol1, String pol2) {
    Polinom polinom1 = creazaPolinom(pol1);
    Polinom polinom2 = creazaPolinom(pol2);
    int i = 0;
    int j = 0;
    ArrayList<Monom> pol = new ArrayList<Monom>();
    while (i < polinom1.getPolinom().size() && j < polinom2.getPolinom().size()) {
        if (polinom1.getPolinom().get(i).getGradMonom() ==
            polinom2.getPolinom().get(j).getGradMonom()) {
            if (polinom1.getPolinom().get(i).getCoeficient() +
                polinom2.getPolinom().get(j).getCoeficient() != 0) {
                pol.add(new Monom(
                    polinom1.getPolinom().get(i).getCoeficient() +
                    polinom2.getPolinom().get(j).getCoeficient(),
                    polinom1.getPolinom().get(i).getGradMonom()));
                i++;
                j++;
            } else {
                if (polinom1.getPolinom().get(i).getGradMonom() >
                    polinom2.getPolinom().get(j).getGradMonom()) {
                    pol.add(new Monom(polinom1.getPolinom().get(i).getCoeficient(),
                        polinom1.getPolinom().get(i).getGradMonom()));
                    i++;
                } else {
                    pol.add(new Monom(polinom2.getPolinom().get(j).getCoeficient(),
                        polinom2.getPolinom().get(j).getGradMonom()));
                    j++;
                }
            }
        }
        while (i < polinom1.getPolinom().size()) {
            pol.add(new Monom(polinom1.getPolinom().get(i).getCoeficient(),
                polinom1.getPolinom().get(i).getGradMonom()));
            i++;
        }
        while (j < polinom2.getPolinom().size()) {
            pol.add(new Monom(polinom2.getPolinom().get(j).getCoeficient(),
                polinom2.getPolinom().get(j).getGradMonom()));
            j++;
        }
        Polinom polinom3 = new Polinom(pol);
        return creazaString(polinom3);
    }
}

```

- toate celelalte functii ( scadere, impartire, inmultire) respecta antetul functiei prezentate mai sus; diferenta se poate observa la metodele integreazaPolinom si deriveazaPolinom deoarece primesc un singur polinom ca argument

Pachetul View contine doar clasa View.

### Clasa View

- este clasa care se ocupa de crearea si design-ul interfetei grafice
- interfata contine 3 TextField-uri si 6 butoane
- doua TextField-uri sunt folosite pentru a introduce datele de intrare, si anume polinoamele, iar unul este folosit pentru a afisa rezultatul operatiei dintre cele doua polinoame
- la apasarea unuia dintre butoane se va afisa rezultatul operatiei selectate
- polinoamele trebuie introduce inaintea selectarii unei operatii, in caz contrar programul nu va putea efectua operatia
- pentru operatiile scadere, impartire, derivare si integrare conteaza ordinea in care sunt introduce polinoamele; pentru derivare si integrare se considera polinomul din primul TextField; pentru scadere se considera polinomul din primul TextField ca fiind deimpartitul, iar cel din al 2 lea TextField va contine impartitorul;
- deoarece la impartire se pot obtine 2 polinoame, cat si rest, se vor afisa amandoua in acelasi TextField;

```
public class View {
    private JFrame frame;
    private JLabel label1;
    private JTextField textfield1;
    private JLabel label2;
    .....
    private JButton buttonPlus;
    private JButton buttonDerivare;
    public View() {
        frame = new JFrame("Calculator de Polinoame");
        frame.setSize(650, 380);
        label1 = new JLabel("Introduceti primul polinom:");
        label1.setBounds(20, 10, 200, 40);
        frame.add(label1);
        textfield1 = new JTextField();
        textfield1.setBounds(20, 50, 340, 30);
        frame.add(textfield1); .....
    }
}
```

- in clasa sunt declarate label-urile, textField-urile si butoanele de care avem nevoie pentru interfata
- in constructorul clasei sunt setate dimensiunile si coordonatele pentru componentele interfetei pe care le si adaugam la frame
- de asemenea avem implementate metode pentru a accesa polinoamele introduce si pentru a afisa rezultatul
- clasa contine metode pentru a adauga un ActionListener pentru fiecare buton; exemplu:

```
public void addPlusButtonActionListener(ActionListener actionListener) {
    buttonPlus.addActionListener(actionListener); }
}
```

Pachetul Controller contine doar clasa Controller.

### Clasa Controller

- este clasa care interactioneaza atat cu modelul cat si cu interfata
- Controllerul actioneaza atat asupra modelului cat si asupra view-ului; acesta controleaza fluxul de date din model si actualizeaza vizualizarea ori de cate ori se schimba datele; totodata pastreaza view-ul si modelul separat
- clasa are doua atribute, unul de tipul View, iar celalalt de tipul Model
- clasa Controller gestioneaza ceea ce se intampla in momentul in care este apasat unul din cele sase butoane;

```
public class Controller {

    private View theView;
    private Model theModel;

    public Controller(View theView, Model theModel) {
        this.theModel = theModel;
        this.theView = theView;
        this.theView.addPlusButtonActionListener(new PlusListener());
    }

    class PlusListener implements ActionListener{
        public void actionPerformed(ActionEvent e) {
            String polinom1 = new String();
            String polinom2 = new String();
            String polinom3 = new String();
            polinom1 = theView.getPolinom1();
            polinom2 = theView.getPolinom2();
            polinom3 = theModel.adunaPolinoame(polinom1, polinom2);
            theView.setRezultat(polinom3);
        }
    }
}
```

### Clasa MVC

- clasa se afla in pachetul MVC
- este clasa in care se afla main-ul proiectului,

```
public class MVC {

    public static void main(String[] args) {
        View theView = new View();
        Model theModel = new Model();
        Controller theController = new Controller(theView,theModel);
    }
}
```

## 5 Rezultate

Rezultatele operatiilor au fost verificate cu ajutorul Junit-ului in clasa JUnitTest din pachetul testing. Am ales cateva exemple relevante pentru fiecare tip de operatie si le-am implementat.

Exemple ale unor metode de testare din clasa JUnitTest:

```
public void testAdunare() {                                ----adunarea a doua polinoame----
    Model test = new Model();
    String output = test.adunaPolinoame("2x+1", "6x+2");
    assertEquals("8x+3",output);
    nrTesteCuSucces++;
}

public void testScadereNrEgale() {                        ----scaderea a doua numere egale ----
    Model test = new Model();
    String output = test.scadePolinoame("2x+1", "2x+1");
    assertEquals("0",output);
    nrTesteCuSucces++;
}

public void testImpartire() {                             ----impartirea a doua polinoame----
    Model test = new Model();
    String output = test.impartePolinoame("6x^3+4x^2-5x+5", "-2x^2-1");
    assertEquals("C:-3x-2 ; R: -8x+3",output);
    nrTesteCuSucces++;
}
```

Runs: 7/7    Errors: 0    Failures: 0

Putem observa in partea stanga, ca toate testele implementate au fost realizate cu success.

```

▼ testing.JUnitTest [Runner: JUnit 4] (0.002 s)
  testAdunare (0.000 s)
  testScadereNrEgale (0.000 s)
  testInmultire (0.000 s)
  testImpartire (0.000 s)
  testIntegrare (0.000 s)
  testScadere (0.000 s)
  testDerivare (0.002 s)

```

## 6 Concluzii

In concluzie , datorita acestui proiect am reusit sa ma familiarizez cu dezvoltarea programelor folosind sablonul Model-View-Controller, dar totodata mi-am imbunatatit cunostiintele despre limbajul de programare java si despre paradigmele OOP.

Proiectul ar putea fi extins; de exemplu ar putea sa suporte operatii mai complexe cu polinome sau ar putea fi inclus in cadrul unui proiect pentru calcule mai complicate si extinse, care ar avea nevoie sa foloseasca si operatii pe polinoame.

## 7.Bibliografie

- <http://users.utcluj.ro/~igiosan/Resources/POO/Lab/10-GUI-I.pdf>
- [http://cadredidactice.ub.ro/sorinpopa/files/2018/10/L1\\_diagrame\\_use\\_case.pdf](http://cadredidactice.ub.ro/sorinpopa/files/2018/10/L1_diagrame_use_case.pdf)