



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

CALCULATOR POLINOMIAL

DOCUMENTATIE

Student: Blajan George Paul

Grupa: 302210

CUPRINS

•Obiectivul temei	3
•Analiza problemei,modelare,scenarii,cazuri de utilizare	3
•Proiectare	4
•Implementare	9
•Rezultate	22
•Concluzii	28
•Bibliografie	28

•Obiectivul temei

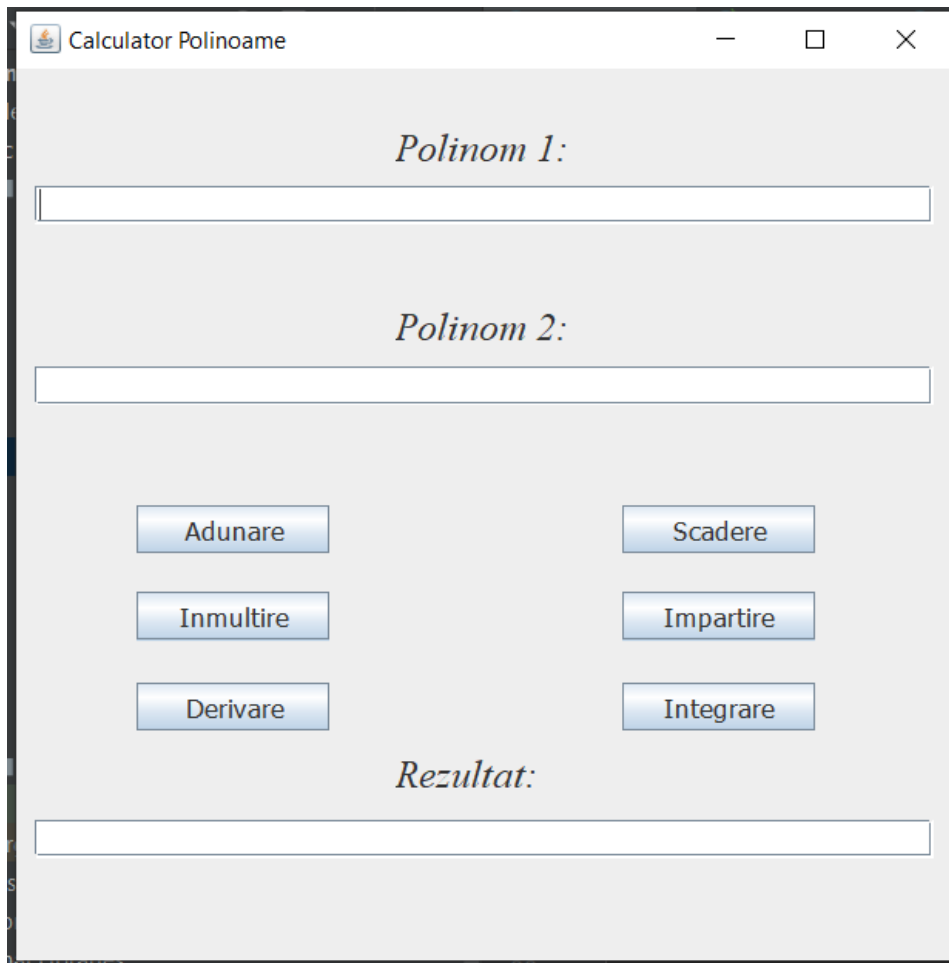
Obiectivul acestei teme este de a dezvolta o aplicatie cu interfata utilizator ce are sa indeplineasca rolul unui calculator ce realizeaza operatii cu polinoame.

Pentru a indeplini obiectivul principal trebuie indeplinite urmatoarele obiective secundare: realizarea interfetei utilizator,implementarea unui mod de transformare a polinoamelor introduse ca tip String in date cu care aplicatia sa poata opera pentru a calcula un rezultat corect,implementarea operatiilor specificate pe polinoame(adunare,scadere,impartire,inmultire,derivare,integrare),debugging,testarea rezultatelor cu JUnit.

•Analiza problemei,modelare,scenarii,cazuri de utilizare

Cadrul de cerinte este implementarea unei aplicatii care sa se comporte ca un calculatori polinomial si sa implementeze operatiile de adunare,scadere,inmultire,impartire,derivare si integrare.

Diagrama use-case pentru a prezenta principiul de functionare al calculatorului este urmatoarea:



Pentru implementarea proiectului s-a optat pentru arhitectura MVC impartita in model,view si controller.

Clasa model se ocupa de implementarea operatiilor pe polinoame si implementeaza doar functionalitati.

Clasa view prezentata mai sus ofera modul de comunicare al aplicatiei cu exteriorul.

Clasa controller este clasa care face legatura intre view si model,controller ul primeste spre exemplu informatii cum ca un buton a fost apasat de la view si ii comanda modelului sa indeplineasca anumite functionalitati.

Clasa exceptie este clasa care se ocupa de tratarea erorilor care pot aparea in timpul rularii aplicatiei.De exemplu introducerea gresita a unui polinom sau anumite cazuri speciale legate de operatii.

Ca structuri de date au fost folosite ArrayList urile pentru stocarea obiectelor de tip monom.

S-au mai folosit Matcher si Pattern,alte doua structuri Java predefinite.

Cele doua au avut rolul de a separa stringul initial reprezentand polinomul in multiple stringuri reprezentand monoamele polinomului. Aceasta impartire a fost facuta cu ajutorul unui regex care semnala elementul folosit in cadrul stringului ca delimitator. De asemenea aceste doua structuri au avut rolul de a valida inputul, fiecare monom este supus unui test cu ajutorul unui alt matcher, pattern si regex iar daca nu corespunde tiparului stabilit clasa exceptie semnaleaza controller ul care opreste executia programului si ofera un mesaj de atentionare.

UML este prescurtarea de la “Unified Modeling Language” si este notația internațională standard pentru analiza și proiectarea orientată pe obiecte.

UML 2.0 defineste treisprezece tipuri de diagrame, împartite în doua categorii:

Diagrame de structură: sase tipuri de diagrame reprezinta structura statica a aplicatiei:

- **Diagrama de clase, diagrame de obiecte**, diagrama de componente, diagrama de structura compozita, diagrama de pachete și diagrama de desfasurare sistematica.

Diagrame de comportament: sapte diagrame ce reprezinta tipuri generale de comportament:

- Diagrama de activitati, diagrama de interactiune, diagrama cazurilor de utilizare (use case), diagrama de secvente, diagrama de stare, diagrama de comunicare, diagrama de timp.

Tipuri de relații:

- Asociere
- Agregare
- Compozitie

- Dependenta
- Generalizare
- Realizare (sau Implementare)

Folosirea asocierilor:

Trei scopuri generale. Pentru a reprezenta:

- O situație în care un obiect de o clasă folosește serviciile unui alt obiect, sau ele își folosesc reciproc serviciile – adică un obiect îi trimite mesaje celuilalt sau își trimit mesaje între ele.
- În primul caz, navigabilitatea poate fi unidirecțională; în cel de al doilea, ea trebuie să fie bidirecțională.
- Agregarea sau compoziția – unde obiecte de o clasă sunt întregi compusi din obiecte de cealaltă clasă ca părți.
- În acest caz, o relație de tip “folosește” este implicit prezentă – întregul folosește părțile pentru a-și îndeplini funcția, iar părțile pot și ele avea nevoie să folosească întregul.
- O situație în care obiectele sunt înrudite, chiar dacă nu schimbă mesaje
- Aceasta se întâmplă de obicei când cel puțin unul dintre obiecte este folosit în esență la stocare de informație.

Relatii: Agregare:

Relație de tip: “are o/un” .

O formă specială de asociere care modelează relația parte– întreg între un agregat (întregul) și părțile sale.

Relatii: Compoziție:

Relație de tip: “este parte a” .

O formă de agregare cu posesiune puternică și durată de viață care coincid .

Părțile nu pot supraviețui fără existența întregului/agregatului.

Acestea fiind spuse diagrama UML reprezentativă pentru proiectul subiect al acestei documentații este următoarea:

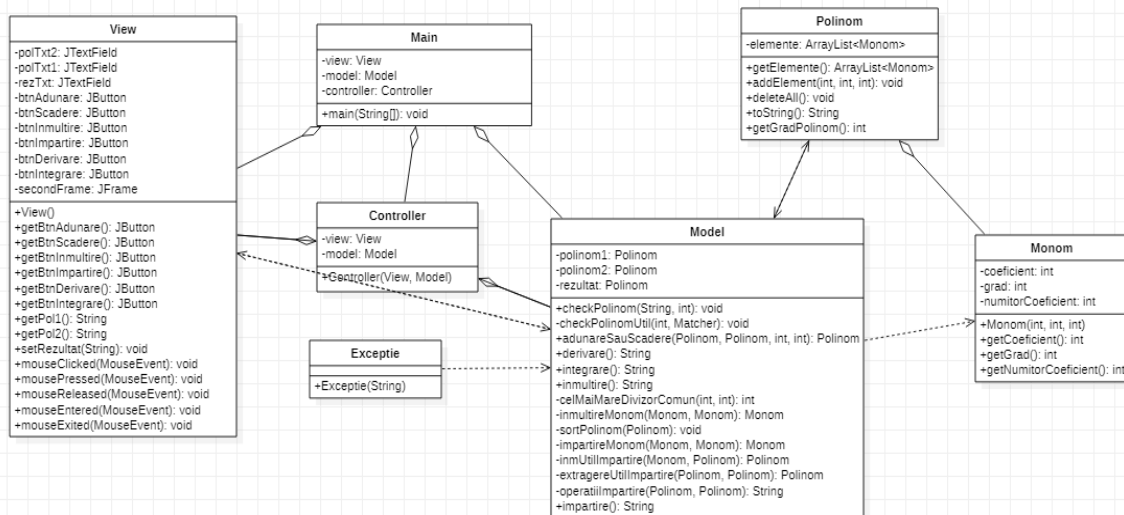


Diagrama este compusa din 7 clase:

- **View:** este clasa responsabila cu interfata grafica si comunicarea cu exteriorul.
- **Model:** aceasta clasa implementeaza toate functionalitatile de care avem nevoie pentru a realiza calculatorul polinomial.
- **Controller:** controller-ul este dupa cum ii spune si numele componenta care comanda actiunile care au loc in functie de ce inregistreaza din partea view-ului, aceste actiuni sunt indeplinite prin intermediul modelului.
- **Main:** este clasa principala a proiectului dare este prima clasa executata de catre compilator, in aceasta clasa n am facut decat sa instantiem obiectele view, model si controller.
- **Polinom:** joaca un rol important in proiect deoarece este reprezentarea OOP a unui polinom, stocheaza prin intermediul structurii de date de tip ArrayList obiecte de tip Monom.
- **Monom:** este transpunerea unui Monom in sintaxa OOP, retine variabile instantia cum ar fi coeficientul monomului, gradul lui si numitorul coeficientului daca este cazul. O multime de obiecte din aceasta clasa alcatuieste un polinom.

- Exceptie: este o clasa de tip util de care ne folosim pentru a atentiona anumite erori in interactiunea cu aplicatia calculator.
- Clasele de tip test: aceste clase nu apar explicit in diagrama UML insa rolul lor este unul important,de a testa parti din proiect intr un mod de sine statator.

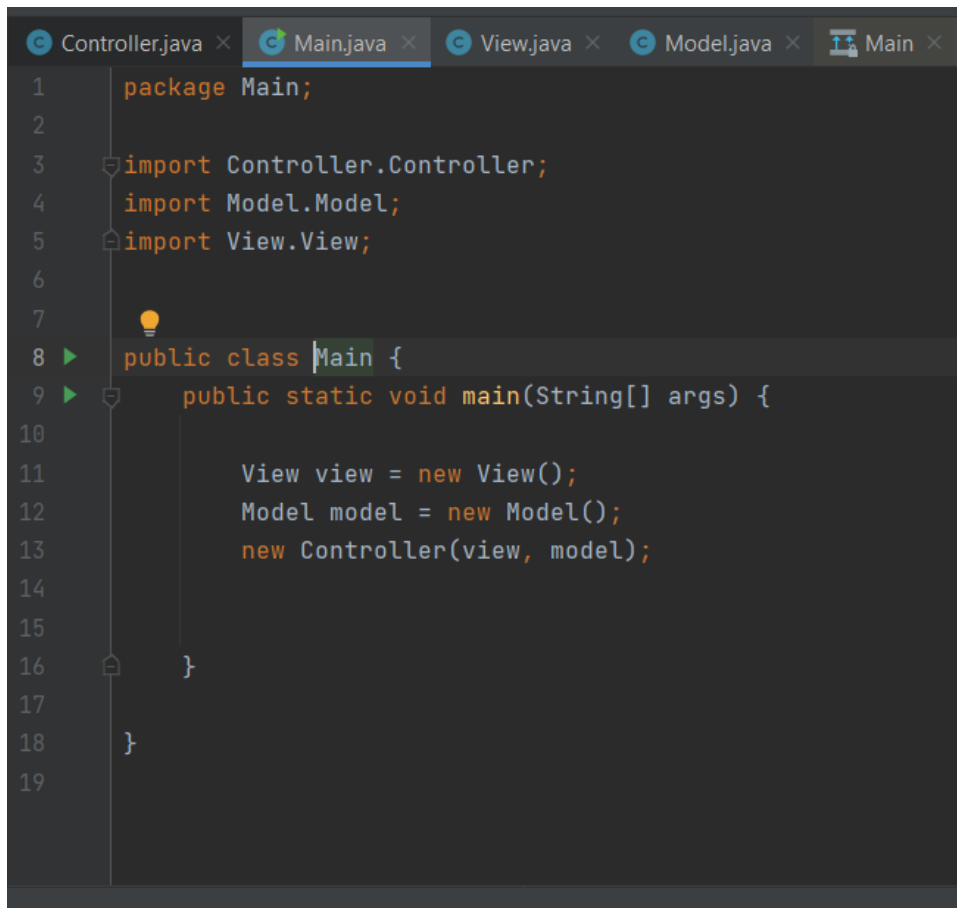
•Implementare

În matematica, un **polinom** este o expresie construita dintr-una sau mai multe variabile si constante, folosind doar operatii de adunare, scadere, înmultire si ridicare la putere constanta pozitiva întreaga. X^2-4X+7 este un polinom.

Polinoamele sunt construite din termeni numiti monoame, care sunt alcatuite dintr-o constanta (numită coeficient) înmultita cu una sau mai multe variabile. Fiecare variabila poate avea un exponent constant întreg pozitiv. Exponentul unei variabile dintr-un monom este egal cu gradul acelei variabile în acel monom.

Clasa Main:

Clasa Main este prima clasa compilata de catre compilator,in aceasta clasa avem 3 variabile instantate , View,Model si Controller. Clasa Main reprezinta punctul de start al proiectului.

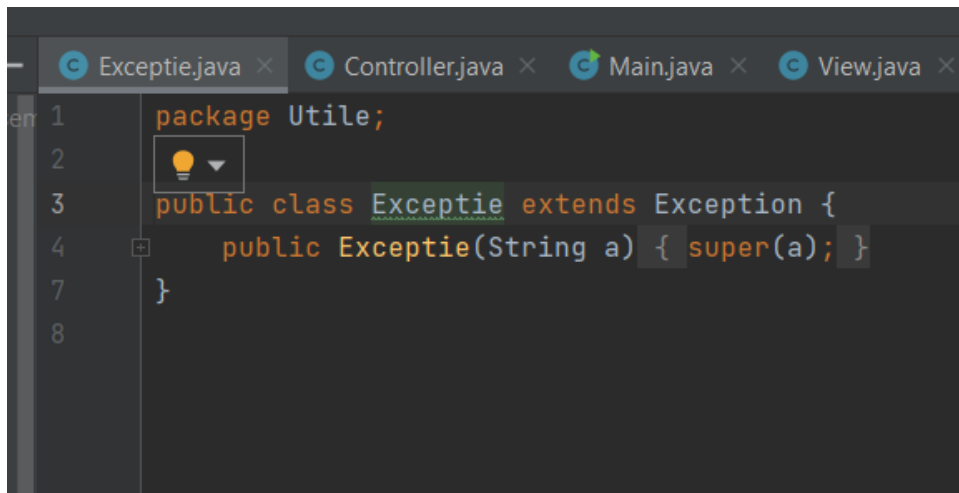


```
1 package Main;
2
3 import Controller.Controller;
4 import Model.Model;
5 import View.View;
6
7
8 public class Main {
9     public static void main(String[] args) {
10
11         View view = new View();
12         Model model = new Model();
13         new Controller(view, model);
14
15     }
16 }
17
18 }
19
```

Clasa Exceptie:

Clasa Exceptie este o clasa de tip util care arunca o exceptie in momentul in care Stringul introdus de catre utilizator nu corespunde cu regex-ul prestabilit ca tipar pentru validarea formei polinomului. Clasa Exceptie extinde Exceptions si este folosita in cadrul clasei Controller pentru a arunca exceptii in momentul in care apare o anumita eroare in rularea aplicatiei.

Are un singur constructor care primeste un string care reprezinta textul ce trebuie afisat in momentul in care clasa JOptionPane afiseaza caseta de dialog indicand eroarea.

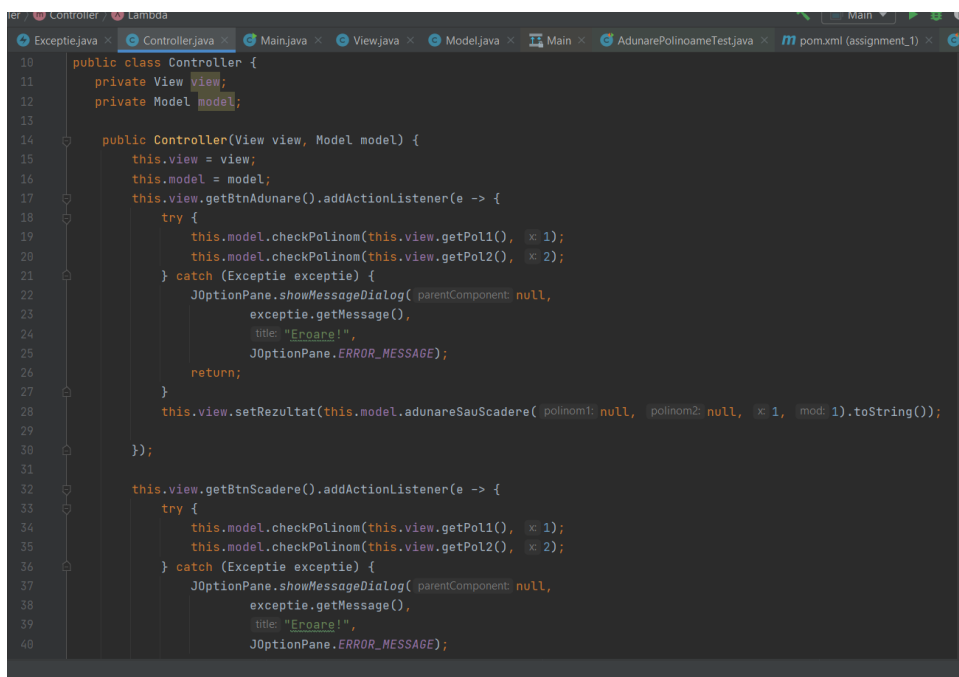


```
1 package Utile;
2
3 public class Exceptie extends Exception {
4     public Exceptie(String a) { super(a); }
5
6
7 }
8
```

Clasa Controller:

Aceasta clasa este responsabila de controlul asupra claselor View si Model, practic acesta este panoul de comanda al proiectului. Clasa Controller are 2 variabile instantia de tip View si Model iar constructorul acestei clase contine instante de ActionListener-i aplicate pointerilor spre butoanele din View, pointeri obtinuti prim metodele de get din clasa View.

Clasa view testeaza tot in interiorul Controller-ului daca polinomul introdus este unul valid apeland metoda checkPolinom din Clasa model, daca aceasta metoda aflata in clasa Model arunca o exceptie clasa Controller o intercepteaza si o trateaza prin afisarea unui mesaj de eroare cu ajutorul clasei JOptionPane.



```
10 public class Controller {
11     private View view;
12     private Model model;
13
14     public Controller(View view, Model model) {
15         this.view = view;
16         this.model = model;
17         this.view.getBtnAdunare().addActionListener(e -> {
18             try {
19                 this.model.checkPolinom(this.view.getPol1(), x 1);
20                 this.model.checkPolinom(this.view.getPol2(), x 2);
21             } catch (Exceptie exceptie) {
22                 JOptionPane.showMessageDialog( parentComponent: null,
23                     exceptie.getMessage(),
24                     title: "Eroare!",
25                     JOptionPane.ERROR_MESSAGE);
26                 return;
27             }
28             this.view.setRezultat(this.model.adunareSauScadere( polinom1: null, polinom2: null, x 1, (mod: 1).toString()));
29         });
30
31     this.view.getBtnScadere().addActionListener(e -> {
32         try {
33             this.model.checkPolinom(this.view.getPol1(), x 1);
34             this.model.checkPolinom(this.view.getPol2(), x 2);
35         } catch (Exceptie exceptie) {
36             JOptionPane.showMessageDialog( parentComponent: null,
37                 exceptie.getMessage(),
38                 title: "Eroare!",
39                 JOptionPane.ERROR_MESSAGE);
40         }
41     });
42 }
```

Controller-ul preia stringurile introduse de utilizator prin interfata View cu ajutorul metodelor `.getPol1`, `.getPol2` si le ofera ca parametrii metodei `checkPolinom` din Model care imparte stringul oferit ipotetic polinom in monoame dupa un regex prestabilit si verifica validitatea fiecarui monom in parte conform altui regex prestabilit. Daca vreun monom nu corespunde metoda `checkPolinom` arunca o exceptie care este interceptata in Controller, daca nu fluxul de comanda merge mai departe si Controller-ul apeleaza metoda care implementeaza operatia corespunatoare butonului apasat in interfata View iar rezultat oferit de metoda operatie o trimite ca parametru de afisat metodei `.setRezultat` din clasa View.

Metoda `.getBtn*` din clasa View returneaza pointer spre butonul selectat iar cu ajutorul acestei metode Controller-ul poate crea un `ActionListener` pe acel buton.

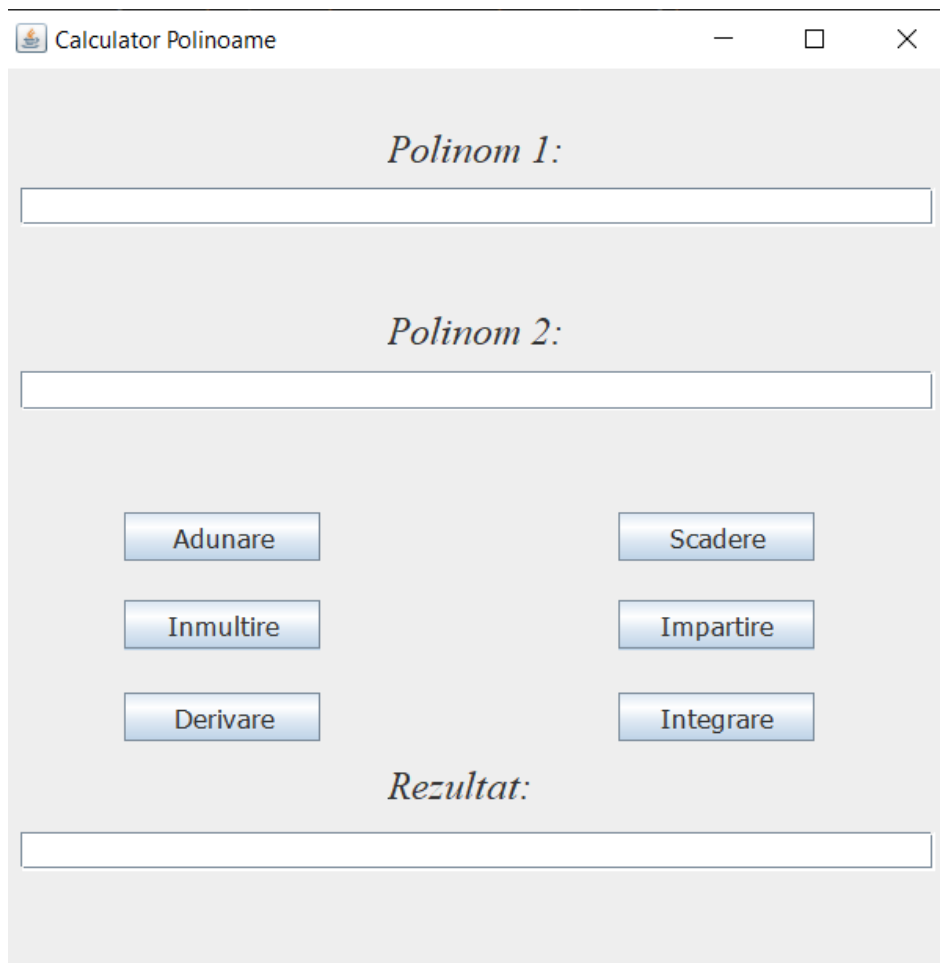
Clasa View:

Clasa View este clasa care creeaza interfata ce interactioneaza cu utilizator si ofera informatii controller-ului despre ce date sunt primite de sistem(butoane apasate,polinoame introduse,etc).

Clasa View importa metode si clase din biblioteca swing pentru a realiza grafica.

Aceasta clasa are 3 `JTextField`uri,6 `JButton`-uri si un `JFrame` declarate ca variabile instantata.Acestea au fost declarate ca instantata si nu locale in constructor deoarece avem nevoie de ele in metodele care returneaza pointeri spre aceste componente ale `JFrame`-ului principal pentru a le putea aplica `ActionListener`eri.

Interfata realizata cu ajutorul clasei View:



Interfata are 6 butoane reprezentand fiecare operatie implementata de calculator si 3 campuri de introducere/afisare a textului.

De asemenea interfata are mesaje de informare cu privire la faptul ca operatiile de integrare/derivare iau in considerare doar valoare polinomului1.

Aceste mesaje de atentionare apare in momentul in care se trece cu mouse-ul peste respectivele butoane.

Aceasta functie a fost implementata cu ajutorul MouseListener-ilor si metodelor mostenite MouseEntered, MouseExited, de asemenea pentru a afisa informatiile s-a folosit un al 2 lea frame cu un Panel negru cu scris alb.

—

□

×

—

□

×

Acesta operatie ia in considerare valoarea polinomului 1 si ignora valoarea polinomului 2.

Polinom 1:

Polinom 2:

Adunare

Inmultire

Derivare

Scadere

Impartire

Integrare

Rezultat:

```
Exceptie.java × Controller.java × Main.java × View.java × Model.java × Main × AdunarePolino
4 import javax.swing.border.EmptyBorder;
5 import javax.swing.border.TitledBorder;
6 import java.awt.*;
7 import java.awt.event.MouseEvent;
8 import java.awt.event.MouseListener;
9
10 public class View implements MouseListener {
11
12     private final JTextField polTxt2;
13     private final JTextField polTxt1;
14     private final JTextField rezTxt;
15     private final JButton btnAdunare;
16     private final JButton btnScadere;
17     private final JButton btnInmultire;
18     private final JButton btnImpartire;
19     private final JButton btnDerivare;
20     private final JButton btnIntegrare;
21     private final JFrame secondFrame = new JFrame();
22
23
24     public View() {
25         JFrame mainFrame = new JFrame( title: "Calculator Polinoame");
26         mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27         mainFrame.setBounds( x: 550, y: 100, width: 500, height: 500);
28         mainFrame.setVisible(true);
29
30         JPanel contentPane = new JPanel();
31         contentPane.setBorder(new EmptyBorder( top: 5, left: 5, bottom: 5, right: 5));
32         mainFrame.setContentPane(contentPane);
33         contentPane.setLayout(null);
34
```

```
Exceptie.java × Controller.java × Main.java × View.java × Model.java × Main × AdunarePolinoameTest.java ×
34
35     polTxt1 = new JTextField();
36     polTxt1.setBounds( x: 10, y: 61, width: 466, height: 20);
37     contentPane.add(polTxt1);
38     polTxt1.setColumns(10);
39
40     JLabel lbl1 = new JLabel( text: "Polinom 1:");
41     lbl1.setFont(new Font( name: "Times New Roman", Font.ITALIC, size: 20));
42     lbl1.setBounds( x: 197, y: 32, width: 96, height: 20);
43     contentPane.add(lbl1);
44
45     JLabel lbl2 = new JLabel( text: "Polinom 2:");
46     lbl2.setFont(new Font( name: "Times New Roman", Font.ITALIC, size: 20));
47     lbl2.setBounds( x: 197, y: 125, width: 96, height: 20);
48     contentPane.add(lbl2);
49
50     polTxt2 = new JTextField();
51     polTxt2.setColumns(10);
52     polTxt2.setBounds( x: 10, y: 155, width: 466, height: 20);
53     contentPane.add(polTxt2);
54
55     btnAdunare = new JButton( text: "Adunare");
56     btnAdunare.setFont(new Font( name: "Tahoma", Font.PLAIN, size: 14));
57     btnAdunare.setBounds( x: 63, y: 227, width: 100, height: 25);
58     contentPane.add(btnAdunare);
59
60     btnScadere = new JButton( text: "Scadere");
61     btnScadere.setFont(new Font( name: "Tahoma", Font.PLAIN, size: 14));
62     btnScadere.setBounds( x: 315, y: 227, width: 100, height: 25);
63     contentPane.add(btnScadere);
64
```

```

View
Exception.java x Controller.java x Main.java x View.java x Model.java x Main x AdunarePolinoameTestJ
64
65     btnInmultire = new JButton( text: "Inmultire");
66     btnInmultire.setFont(new Font( name: "Tahoma", Font.PLAIN, size: 14));
67     btnInmultire.setBounds( x: 63, y: 272, width: 100, height: 25);
68     contentPane.add(btnInmultire);
69
70     btnImpartire = new JButton( text: "Impartire");
71     btnImpartire.setFont(new Font( name: "Tahoma", Font.PLAIN, size: 14));
72     btnImpartire.setBounds( x: 315, y: 272, width: 100, height: 25);
73     contentPane.add(btnImpartire);
74
75     btnDerivare = new JButton( text: "Derivare");
76     btnDerivare.setFont(new Font( name: "Tahoma", Font.PLAIN, size: 14));
77     btnDerivare.setBounds( x: 63, y: 319, width: 100, height: 25);
78     contentPane.add(btnDerivare);
79
80     btnIntegrare = new JButton( text: "Integrare");
81     btnIntegrare.setFont(new Font( name: "Tahoma", Font.PLAIN, size: 14));
82     btnIntegrare.setBounds( x: 315, y: 319, width: 100, height: 25);
83     contentPane.add(btnIntegrare);
84
85     JLabel lblRezultat = new JLabel( text: "Rezultat:");
86     lblRezultat.setFont(new Font( name: "Times New Roman", Font.ITALIC, size: 20));
87     lblRezultat.setBounds( x: 197, y: 357, width: 96, height: 20);
88     contentPane.add(lblRezultat);
89
90     rezTxt = new JTextField();
91     rezTxt.setColumns(10);
92     rezTxt.setBounds( x: 10, y: 390, width: 466, height: 20);
93     contentPane.add(rezTxt);
94

```

```

94
95     this.btnDerivare.addMouseListener( this);
96     this.btnIntegrare.addMouseListener( this);
97
98     secondFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
99     secondFrame.setBounds( x: 100, y: 100, width: 247, height: 125);
100     secondFrame.setLocation(mainFrame.getLocation());
101     JPanel contentPane2 = new JPanel();
102     contentPane2.setBackground(Color.BLACK);
103     contentPane2.setBorder(new TitledBorder( border: null, title: "", TitledBorder.LEADING, TitledBorder.TOP, titleFont: null, titleColor: null));
104     secondFrame.setContentPane(contentPane2);
105     contentPane2.setLayout(null);
106
107     JLabel lblNewLabel = new JLabel( text: "<html>Acesta operatie ia in considerare <br>valoarea polinomului 1 si ignora<br>valoarea p");
108     lblNewLabel.setForeground(Color.WHITE);
109     lblNewLabel.setFont(new Font( name: "Times New Roman", Font.ITALIC, size: 15));
110     lblNewLabel.setBounds( x: 10, y: 0, width: 210, height: 91);
111     contentPane2.add(lblNewLabel);
112
113
114 }
115
116 public JButton getBtnAdunare() { return this.btnAdunare; }
117
118 public JButton getBtnScadere() { return this.btnScadere; }
119
120 public JButton getBtnInmultire() { return this.btnInmultire; }
121
122 public JButton getBtnImpartire() { return this.btnImpartire; }
123
124 public JButton getBtnDerivare() { return this.btnDerivare; }
125

```

Clasa Model:

Clasa Model este cea mai complexa clasa a acestui proiect. Aceasta clasa implementeaza toate functionalitatile prezente in proiect. Implementeaza operatiile si metodele auxiliare necesare operatiilor.

Clasa Model declara 3 variabile instantate de tip Polinom, Polinom1, Polinom2 si Rezultat pentru a stoca in Polinom1 si Polinom2 datele introduse de utilizator iar in Rezultat rezultatul in urma operatiilor pentru a-l transmite Controller-ului.

Clasa model implementeaza metoda checkPolinom care cu ajutorul metodei auxiliare checkPolinomUtil imparte stringul polinomial in stringuri monomiale iar apoi le testeaza validand sau invalidand corectitudinea tiparului.

Pentru a fi impartit polinomul in mai multe monoame au fost folosite urmatoarele Matcher si Pattern:

```
Pattern pattern1 = Pattern.compile("[+-]?[^-+]+");  
Matcher matcher1 = pattern1.matcher(a);
```

La fiecare apelare a metodei checkPolinom polinoamele interne ale clasei sunt resetate pentru a evita appendarea la un continut vechi rezultand un continut eronat:

```
public void checkPolinom(String a, int x) throws Exceptie {  
    if ((a.equals(" ")) || (a.equals("")))  
        throw new Exceptie(a: "Utile.Polinom invalid");  
    if (x == 1) {  
        this.polinom1.deleteAll();  
        this.polinom2.deleteAll();  
        this.rezultat.deleteAll();  
    }  
}
```

In urma impartirii in monoame acestea se transmit mai departe metodei auxiliare pentru a fi testata validitatea cu ajutorul regexului:

```
Pattern pattern2 = Pattern.compile("[+-]?[0-9]*([Xx])?\\^?[0-9]*");  
Matcher matcher2;
```

Daca monomul trece testul de validare atunci se incepe prelucrarea acestuia, se parcurge string-ul pentru a se obtine coeficientul si gradul si a le transforma in date de tip int pentru a se putea opera cu ele:

```

private void checkPolinomUtil(int x, Matcher matcher1) {
    int i = 0;
    int validareGrad = 0;
    int grad = 0;
    int coefficient = 0;
    int operatie = 1;
    while ((i < matcher1.group(1).length()) && (matcher1.group(1).charAt(i) != 'x') && (matcher1.group(1).charAt(i) != 'x')) {
        if ((matcher1.group(1).charAt(i) >= '0') && (matcher1.group(1).charAt(i) <= '9')) {
            if (i != 0) {
                if (matcher1.group(1).charAt(i - 1) == '-')
                    operatie = 0;
            }
            if (operatie == 1)
                coefficient = coefficient * 10 + (matcher1.group(1).charAt(i) - 48);
            else
                coefficient = coefficient * 10 - (matcher1.group(1).charAt(i) - 48);
        }
        i++;
    }

    while (i < matcher1.group(1).length()) {
        if ((matcher1.group(1).charAt(i) == 'x') || (matcher1.group(1).charAt(i) == 'x'))
            validareGrad = 1;
        if ((matcher1.group(1).charAt(i) >= '0') && (matcher1.group(1).charAt(i) <= '9'))
            grad = grad * 10 + (matcher1.group(1).charAt(i) - 48);
    }
}

```

Iar la final se adauga acest coeficient si grad unuia dintre polinoamele interne ale clasei prin metoda .addElement aflata in clasa Polinom.

Clasa Model implementeaza operatiile de adunare si scadere in aceeasi metoda,folosindu-se de o variabila parametru pentru a sti daca se face adunare sau scadere.

Se parcurg cele 2 array uri ale polinoamelor intre care se face adunarea si daca se gasesc 2 termeni de grad egal li se aduna coeficientii iar acel monom se adauga polinomului rezultat.

```

public Polinom adunareSauScadere(Polinom polinom1, Polinom polinom2, int x, int mod) {
    int indexPol1 = 0;
    int indexPol2 = 0;
    if (mod == 1) {
        polinom1 = this.polinom1;
        polinom2 = this.polinom2;
    }

    while ((indexPol1 < polinom1.get Elemente().size()) && (indexPol2 < polinom2.get Elemente().size())) {
        if (polinom1.get Elemente().get(indexPol1).getGrad() > polinom2.get Elemente().get(indexPol2).getGrad()) {
            rezultat.addElement(polinom1.get Elemente().get(indexPol1).getCoefficient(), polinom1.get Elemente().get(indexPol1).getGrad(),
                                indexPol1++);
        } else {
            if (polinom1.get Elemente().get(indexPol1).getGrad() < polinom2.get Elemente().get(indexPol2).getGrad()) {
                if (x == 1)
                    rezultat.addElement(polinom2.get Elemente().get(indexPol2).getCoefficient(), polinom2.get Elemente().get(indexPol2).getGrad(),
                                        indexPol2++);
                else
                    rezultat.addElement(-polinom2.get Elemente().get(indexPol2).getCoefficient(), polinom2.get Elemente().get(indexPol2).getGrad(),
                                        indexPol2++);
            } else {
                if (polinom1.get Elemente().get(indexPol1).getGrad() == polinom2.get Elemente().get(indexPol2).getGrad()) {
                    if (x == 1)
                        rezultat.addElement(coefficient: polinom1.get Elemente().get(indexPol1).getCoefficient() + polinom2.get Elemente().get(indexPol2).getCoefficient(),
                                            indexPol1++,
                                            indexPol2++);
                    else
                        rezultat.addElement(coefficient: polinom1.get Elemente().get(indexPol1).getCoefficient() - polinom2.get Elemente().get(indexPol2).getCoefficient(),
                                            indexPol1++,
                                            indexPol2++);
                }
            }
        }
    }
}

```

Pentru operatia de derivare se parcurge Polinom1 iar coeficientii devin vechiul coeficient*vechiul grad al monomului iar noul grad devine vechiul grad-1.

```

public String derivare() {
    for (Monom i : polinom1.get Elemente())
        if (i.getGrad() > 0)
            rezultat.addElement( coefficient: i.getCoefficient() * i.getGrad(), grad: i.getGrad() - 1, numitorCoefficient: 1);

    return rezultat.toString();
}

public String integrare() {...}

```

Pentru operatia de integrare se parcurge Polinomul 1 iar coeficientii sunt impartiti la cmmdc dintre ei si numitor care este gradul vechi al monomului plus 1 iar gradul monomului creste cu 1.

Pentru a rezolva problema coeficientilor intregi am adaugat o variabila numitor pentru a putea reprezenta fractii la reprezentarea polinoamelor.

```

public String integrare() {
    for (Monom i : polinom1.get Elemente())
        rezultat.addElement( coefficient: i.getCoefficient() / celMaiMareDivizorComun(i.getCoefficient(), i.getGrad() + 1), grad: i.getGrad() + 1, numitorCoefficient: i.getGrad() + 1);

    return rezultat.toString();
}

```

Operatia de inmultire se realizeaza inmultind fiecare monom cu fiecare alt monom al celuilalt.

```

public String inmultire() {
    Polinom aux = new Polinom();
    int coefficient;
    int grad;
    int ok;
    ArrayList<Integer> puteri = new ArrayList<>();

    for (Monom i : polinom1.get Elemente()) {
        for (Monom j : polinom2.get Elemente())
            aux.addElement(inmultireMonom(i, j).getCoefficient(), inmultireMonom(i, j).getGrad(), inmultireMonom(i, j).getNumitorCoefficient());
    }

    for (Monom i : aux.get Elemente()) {
        coefficient = i.getCoefficient();
        grad = i.getGrad();
        ok = 1;
        for (Integer o : puteri)
            if (o == grad) {
                ok = 0;
                break;
            }
        if (ok == 1) {
            puteri.add(grad);
            for (Monom j : aux.get Elemente())
                if (grad == j.getGrad()) {
                    if (aux.get Elemente().indexOf(j) != aux.get Elemente().indexOf(i))
                        coefficient += j.getCoefficient();
                }
            rezultat.addElement(coefficient, grad, numitorCoefficient: 1);
        }
    }
}

```

Operatia de impartire are 3 pasi:

Impartirea monomului cu gradul cel mai mare al deimpartitului la monomul cu gradul cel mai mare al impartitorului.

Inmultirea catului cu impartitorul.

Scaderea rezultatului anterior din deimpartit pentru a obtine noua forma a deimpartitului.

```

283     polinomCat.addElement(aux.getCoefficient(), aux.getGrad(), aux.getNumitorCoefficient());
284     a = extragereUtilImpartire(a,
285         inmUtilImpartire(a,
286             inmUtilImpartire(aux, b));
287     if(a.getElemente().isEmpty()) {
288         break;
289     }
290 }
291 return "Cat: " + polinomCat.toString() + " Rest: " + a.toString();
292 }
293
294 public String impartire() {
295     sortPolinom(polinom1);
296     sortPolinom(polinom2);
297     if(polinom1.getGradPolinom()==0 && polinom2.getGradPolinom()==0) {
298         float s1 = (float) polinom1.getElemente().get(0).getCoefficient() / polinom2.getElemente().get(0).getCoefficient();
299         int s2 = polinom1.getElemente().get(0).getCoefficient() / polinom2.getElemente().get(0).getCoefficient();
300         if (s1 != s2) {
301             JOptionPane.showMessageDialog(parentComponent, null,
302                 "Rezultatul are coeficienti reali",
303                 "Eroare!",
304                 JOptionPane.ERROR_MESSAGE);
305             return "";
306         } else
307             return "Cat: " + polinom1.getElemente().get(0).getCoefficient() / polinom2.getElemente().get(0).getCoefficient() + " Rest: 0";
308     } else {
309         if (polinom1.getGradPolinom() >= polinom2.getGradPolinom())
310             return operatiiImpartire(polinom1, polinom2);
311         else
312             return operatiiImpartire(polinom2, polinom1);
313     }
}

```

```

307 public String inmultire() {...}
308
309 private int celMaiMareDivizorComun(int x1, int x2) {...}
310
311 private Monom inmultireMonom(Monom a, Monom b) {...}
312
313 private void sortPolinom(Polinom a) {...}
314
315 private Monom impartireMonom(Monom a, Monom b) {...}
316
317 private Polinom inmUtilImpartire(Monom a, Polinom b) {...}
318
319 private Polinom extragereUtilImpartire(Polinom a, Polinom b) {...}
320
321 private String operatiiImpartire(Polinom a, Polinom b) {
322     Polinom polinomCat = new Polinom();
323     while ((a.getGradPolinom() >= b.getGradPolinom()) && (a != a.getElemente().get(0).getCoefficient())) {
324         float s1 = (float) a.getElemente().get(0).getCoefficient() / b.getElemente().get(0).getCoefficient();
325         int s2 = a.getElemente().get(0).getCoefficient() / b.getElemente().get(0).getCoefficient();
326         if (s1 != s2) {...}
327         Monom aux = impartireMonom(a.getElemente().get(0), b.getElemente().get(0));
328         polinomCat.addElement(aux.getCoefficient(), aux.getGrad(), aux.getNumitorCoefficient());
329         a = extragereUtilImpartire(a,
330             inmUtilImpartire(aux, b));
331         if(a.getElemente().isEmpty()) {
332             break;
333         }
334     }
335     return "Cat: " + polinomCat.toString() + " Rest: " + a.toString();
336 }

```

Clasa Monom:

Aceasta clasa stocheaza coeficientul, gradul si numitorul coeficientului unui monom.

Implementeaza metode de get pentru a facilita accesul la grade si coeficienti.

```

1 package Util;
2
3 public class Monom {
4     private final int coeficient;
5     private final int grad;
6     private final int numitorCoeficient;
7
8     public Monom(int coeficient, int grad, int numitorCoeficient) {
9         this.coeficient = coeficient;
10        this.grad = grad;
11        this.numitorCoeficient = numitorCoeficient;
12    }
13
14    public int getCoefficient() { return this.coeficient; }
17
18    public int getGrad() { return this.grad; }
21
22    public int getNumitorCoefficient() { return this.numitorCoefficient; }
25
26 }

```

Clasa Polinom:

Aceasta clasa implementeaza subiectul acestui proiect, polinoamele.

Stocheaza intr-un ArrayList o multime de obiecte monoame alcatuind un polinom.

Are metode de delete, get lista de monoame si toString.

```

public class Polinom {
    private final ArrayList<Monom> elemente = new ArrayList<>();

    public ArrayList<Monom> getElemente() { return this.elemente; }

    public void addElement(int coeficient, int grad, int numitorCoefficient) {
        this.elemente.add(new Monom(coeficient, grad, numitorCoefficient));
    }

    public void deleteAll() { this.elemente.clear(); }

    public String toString() {
        StringBuilder s = new StringBuilder();
        if (this.elemente.size() == 0)
            s = new StringBuilder("0");
        else {
            for (Monom i : this.elemente)
                if (i.getCoefficient() != 0) {
                    if (i.getGrad() == 0) {
                        if (i.getCoefficient() < 0) {
                            if (i.getNumitorCoefficient() != 1)
                                s.append(i.getCoefficient()).append("/").append(i.getNumitorCoefficient());
                            else
                                s.append(+i.getCoefficient());
                        } else {
                            if (i.getGrad() == this.elemente.get(0).getGrad()) {
                                if (i.getNumitorCoefficient() != 1)
                                    s.append(i.getCoefficient()).append("/").append(i.getNumitorCoefficient());
                                else

```

```
Excepie.java × Controller.java × Main.java × View.java × Monom.java × Polinom.java × AdunarePolinoameTest.java × pom.xml (assignment_1) × DerivareP...
1 if (i.getGrad() == this.elemente.get(0).getGrad()) {
2     if (i.getNumitorCoefficient() != 1)
3         s.append(i.getCoefficient()).append("/").append(i.getNumitorCoefficient());
4     else
5         s.append(i.getCoefficient());
6 } else {
7     if (i.getNumitorCoefficient() != 1)
8         s.append("+").append(i.getCoefficient()).append("/").append(i.getNumitorCoefficient());
9     else
10        s.append("+").append(i.getCoefficient());
11 }
12 }
13 if (i.getGrad() == 1) {
14     if (i.getCoefficient() == 1) {
15         if (i.getGrad() == this.elemente.get(0).getGrad()) {
16             if (i.getNumitorCoefficient() != 1)
17                 s.append(i.getCoefficient()).append("/").append(i.getNumitorCoefficient()).append("X");
18             else
19                 s.append("X");
20         } else {
21             if (i.getNumitorCoefficient() != 1)
22                 s.append("+").append(i.getCoefficient()).append("/").append(i.getNumitorCoefficient()).append("X");
23             else
24                 s.append("+ " + "X");
25         }
26     } else {
27         if (i.getCoefficient() < 0) {
28             if (i.getNumitorCoefficient() != 1)
29                 s.append(i.getCoefficient()).append("/").append(i.getNumitorCoefficient()).append("X");
30             else if (i.getCoefficient() == -1)
31                 s.append("-X");
32         }
33     }
34 }
```

•Rezultate

Scenariile de testare au fost implementate pentru acest proiect cu JUnit.

Pentru fiecare operatie in parte s-a creat o clasa de testare separata.

```
Excepie.java × Controller.java × Main.java × View.java × Monom.java × Polinom.java × AdunarePolinoameTest.java × pom.xml (assignment_1) × DerivareP...
1 import Model.Model;
2 import Utilite.Polinom;
3 import org.junit.jupiter.api.Test;
4
5 import static org.junit.jupiter.api.Assertions.assertEquals;
6
7 public class AdunarePolinoameTest {
8     @Test
9     public void adunarePolinoame() {
10         Model model=new Model();
11         Polinom pol1=new Polinom();
12         Polinom pol2=new Polinom();
13         pol1.addElement( coefficient: 2, grad: 2, numitorCoefficient: 1);
14         pol1.addElement( coefficient: 1, grad: 1, numitorCoefficient: 4);
15         pol1.addElement( coefficient: 3, grad: 0, numitorCoefficient: 1);
16         pol2.addElement( coefficient: 4, grad: 1, numitorCoefficient: 1);
17         pol2.addElement( coefficient: 1, grad: 0, numitorCoefficient: 1);
18         assertEquals(model.adunareSauScadere(pol1, pol2, 1, mod: 0).toString(), actual: "2X^2+5X+4");
19     }
20 }
21 }
```

```

1  import Model.Model;
2  import Utile.Exceptie;
3  import org.junit.jupiter.api.Test;
4
5  import static org.junit.jupiter.api.Assertions.assertEquals;
6
7  public class DerivarePolinoame {
8
9
10
11     @Test
12     public void derivare() throws Exceptie { Model model=new Model();
13         model.checkPolinom( a: "2X^2+X+3", x: 1);
14         assertEquals(model.derivare(), actual: "4X+1");
15     }
16 }
17

```

```

1  import Model.Model;
2  import Utile.Exceptie;
3  import org.junit.jupiter.api.Test;
4
5  import static org.junit.jupiter.api.Assertions.assertEquals;
6
7  public class ImpartirePolinoameTest {
8
9
10
11     @Test
12     public void impartirePolinoame() throws Exceptie { Model model=new Model();
13         model.checkPolinom( a: "X^3-2X^2+6X-5", x: 1);
14         model.checkPolinom( a: "X^2-1", x: 2);
15         assertEquals(model.impartire(), actual: "Cat: X-2 Rest: 7X-7");
16     }
17 }
18

```

```

1  import Model.Model;
2  import Utile.Exceptie;
3  import org.junit.jupiter.api.Test;
4
5  import static org.junit.jupiter.api.Assertions.assertEquals;
6
7  public class InmultirePolinoameTest {
8
9     @Test
10     public void inmultirePolinoame() throws Exceptie {
11         Model model=new Model();
12         model.checkPolinom( a: "3X^2-X+1", x: 1);
13         model.checkPolinom( a: "X-2", x: 2);
14         assertEquals(model.inmultire(), actual: "3X^3-7X^2+3X-2");
15     }
16 }
17

```

```

1 import Model.Model;
2 import Utile.Exceptie;
3 import org.junit.jupiter.api.Test;
4
5 import static org.junit.jupiter.api.Assertions.assertEquals;
6
7 public class IntegrarePolinoameTest {
8     @Test
9     public void integrarePolinoame() throws Exceptie { Model model=new Model();
10         model.checkPolinom( a: "2X^2+X+3", x: 1);
11         assertEquals(model.integrare(), actual: "2/3X^3+1/2X^2+3X");
12     }
13 }

```

```

1 import Model.Model;
2 import Utile.Polinom;
3 import org.junit.jupiter.api.Test;
4
5 import static org.junit.jupiter.api.Assertions.assertEquals;
6
7 public class ScaderePolinoameTest {
8
9     @Test
10     public void scaderePolinoame()
11     {
12         Model model=new Model();
13         Polinom pol1=new Polinom();
14         Polinom pol2=new Polinom();
15         pol1.addElement( coefficient: 2, grad: 2, numitorCoefficient: 1);
16         pol1.addElement( coefficient: 1, grad: 1, numitorCoefficient: 4);
17         pol1.addElement( coefficient: 3, grad: 0, numitorCoefficient: 1);
18         pol2.addElement( coefficient: 4, grad: 1, numitorCoefficient: 1);
19         pol2.addElement( coefficient: 1, grad: 0, numitorCoefficient: 1);
20         assertEquals(model.adunareSauScadere(pol1, pol2, x: 0, mod: 0).toString(), actual: "2X^2-3X+2");
21     }
22 }

```

Se poate observa ca toate testele au avut succes in poza urmatoare:

```

Run: Main x All in assignment_1 x
[Icons] Tests passed: 6 of 6 tests - 40 ms
C:\Program Files\Java\jdk-15.0.2\bin\java.exe" ...
Process finished with exit code 0
<default package> 40 ms
> AdunarePolinoameTest 18 ms
> DerivarePolinoame 2 ms
> ImpartirePolinoameTest 16 ms
> InmultirePolinoameTest 2 ms
> IntegrarePolinoameTest 1 ms
> ScaderePolinoameTest 1 ms
  > scaderePolinoame() 1 ms

```

Alte cazuri exceptionale ar fi operatii pe polinoame cu coeficienti sau grade>9:

Polinom 1:

100X²⁰⁹+13X²+41

Polinom 2:

X¹²+30

Adunare

Scadere

Inmultire

Impartire

Derivare

Integrare

Rezultat:

100X²⁰⁹+X¹²+13X²+71

Polinom 1:

100X²⁰⁹+13X²+41

Polinom 2:

X¹²+30

Adunare

Scadere

Inmultire

Impartire

Derivare

Integrare

Rezultat:

20900X²⁰⁸+26X

Calculator Polinoame

Polinom 1:

100X²⁰⁹+13X²+41

Polinom 2:

X¹²+30

Adunare Scadere

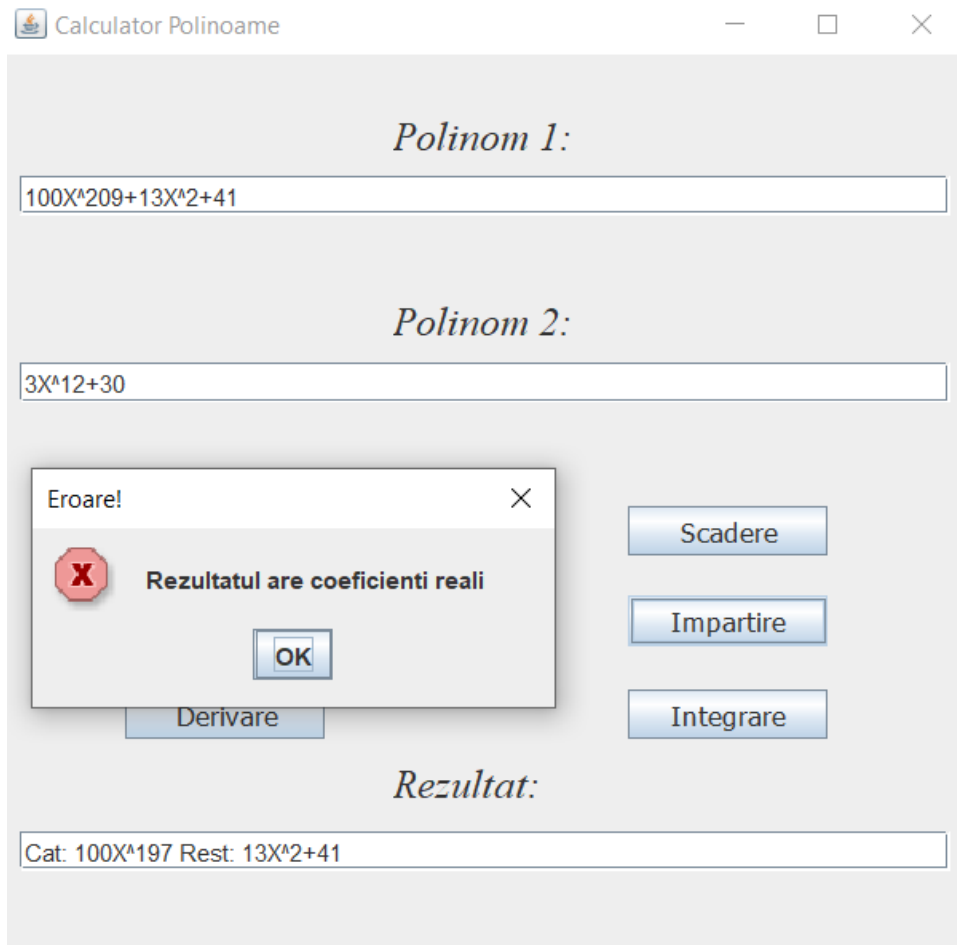
Inmultire Impartire

Derivare Integrare

Rezultat:

Cat: 100X¹⁹⁷ Rest: 13X²+41

O alta situatie exceptionala este cea a integrarii sau impartirii unde putem obtine coeficienti reali, cerinta specificand faptul ca coeficientii stocheaza intregi.



Am tratat acest caz de exceptie prin afisarea unui mesaj.

•Concluzii

In urma acestui proiect principala abilitate care a fost solicitata a fost cea de a putea face debugging.

De asemenea acest proiect a solicitat si o gandire matematica si o vedere logica asupra metodelor de programare pe notiuni matematice.

O dezvoltare ulterioara ar putea fi implementarea acestui calculator pentru a functiona cu coeficienti reali si complecsi.

•Bibliografie

Wikipedia - [Wikipedia](#)

Site-ul domnului profesor Ionel Giosan - [Ion Giosan | Programare Orientată pe Obiecte \(utcluj.ro\)](#)

Stack Overflow - [Stack Overflow - Where Developers Learn, Share, & Build Careers](#)