

Documentație

Sistem de procesare a polinoamelor de o singură variabilă

Student: Costea Ovidiu-Bogdan

Grupa:30229

Contents

1.Obiectivul temei:.....	3
2.Analiza problemei ,modelare,cazuri de utilizare	3
2.1Analiza problemei	3
2.2. Modelarea.....	3
2.3.Cazuri de utilizare.....	3
3.Proiectare.....	4
3.1. Structuri de date.....	4
3.2. Clase	5
3.3Algoritmi folosiți	5
3.4. Pachete	6
3.5. Interfețe	6
3.6.Interfața cu utilizatorul	6
4. Implementare și Testare.....	8
5.Rezultat	15
6.Concluzie	16
7.Bibliografie	16

1. Obiectivul temei:

Principalul scop al acestei teme a fost dezvoltarea unei aplicații, în limbajul de programare Java, capabile să realizeze operații cu polinoame de o singură variabilă și cu coeficienți întregi. Acest software ar trebui să efectueze operații ca adunarea, scăderea, înmulțirea, împărțirea a două polinoame. Alte operații adiționale ar fi derivarea și integrarea.

Proiectul realizat de mine ar trebui să rezolve toate aceste operații menționate mai sus, are o interfață prietenoasă implementat cu ajutorul pachetelor Swing și AWT.

2. Analiza problemei, modelare, cazuri de utilizare

2.1 Analiza problemei

Printr-o scurtă analiză a problemei putem observa componentele de care avem nevoie în elaborarea acestei aplicații. Pe lângă clasele folosite pentru reprezentarea polinomului vom crea o clasă pentru: GUI (Graphical User Interface), controllerul folosit în design patternul MVC și o ultimă clasă în care vom apela componentele care fac parte din MVC.

2.2. Modelarea

De asemenea putem observa că un polinom este format dintr-o listă de monoame. Fiecare monom la rândul său este reprezentat de două câmpuri: unul pentru grad, iar celălalt pentru coeficient. Ne reiese faptul că monomul este entitatea de bază utilizată în polinom, care are un set de câmpuri descrise înainte. Astfel nu mai trebuie să căutăm să dividem polinomul în părți mai mici.

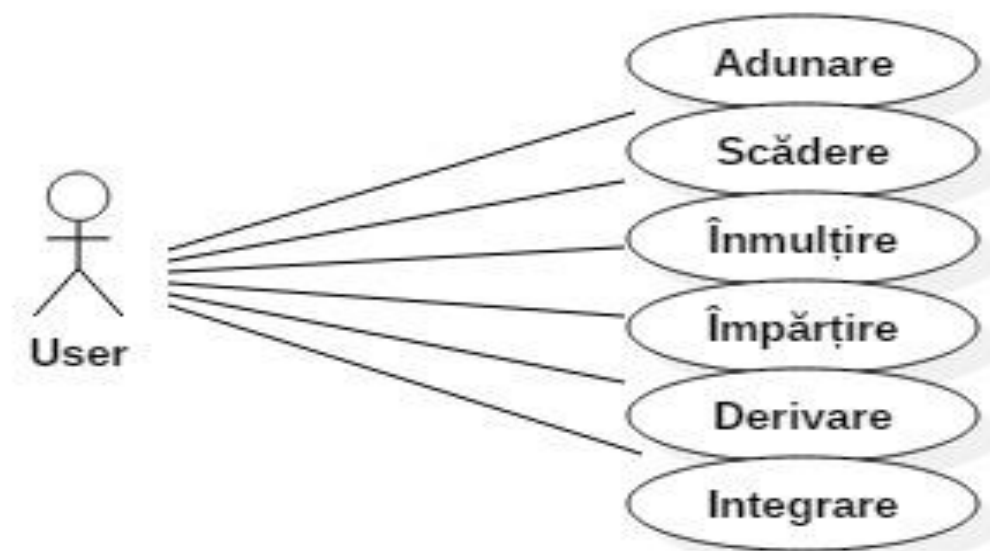
2.3. Cazuri de utilizare

În primul rând utilizatorul trebuie să introducă polinomul/polinoamele, sub forma unui sir de caractere fără spații între termeni sau alte forme de monoame care nu sunt acceptate în formatul cu care am lucrat ("...+coeficient(întreg) $X^{\text{putere(întreg)}}$ "). Dacă în cazul în care utilizatorul introduce alte caractere decât cifrele de la 0-9, X , $+$, $-$, $^$ va primi o înștiințare asupra greșelilor făcute în scrierea polinomului.

O dată ce datele au fost introduse corect trebuie setată operația pe care dorim să o efectuăm asupra polinomului/polinoamelor introduse. Operațiile vor fi vizibile prin simpla apăsare a ComboBoxului. Următorul pas după selectarea operației este calcularea rezultatului care va apărea în TextArea în cazul în care s-au urmat cu strictețe pașii enumerați înainte și după apăsarea butonului "Calculează".

Ca exemplu dacă dorim să calculăm înmulțirea a două polinoame " X^1+1 ", " X^1-1 " ar trebui ca zona textArea să fie actualizată cu valoarea " X^2-1 ". Iar în cazul operației de scădere rezultatul ar trebui să fie " $+2X^2$ ".

Diagrama USE-CASE:



3. Proiectare

Aplicatia a fost dezvoltată printr-o abordare top-down , sistemul fiind proiectat ca un tot, apoi acesta a fost divizat in subsisteme. Clasele au fost realizate intr-un mod abstract, acestea asteptandu-se la un rezultat specific fara a furniza instructiuni de returnare a acelui rezultat. Dupa ce un model acceptabil a fost atins metodele au putut fi definite.

Motivul utilizarii acestei abordari a fost faptul că oferă un foarte bun nivel de abstractizare, acest lucru îmbunătățind viteza de dezvoltare a aplicației, un cod mai ușor de întreținut și mai puțin predispus la erori.

3.1. Structuri de date

Deoarece numarul de monoame conținut de fiecare polinom nu este cunoscut ar fi o decizie greșită dacă am folosi un simplu array deoarece , numărul de monoame poate sa nu se potrivească cu mărimea array-ului ceea ce ar provoca trecere peste limita impusă , astfel numarul de monoame ar fi restricționat ceea ce ar face ca un polinom cu multe monoame să nu poată fi reținut.

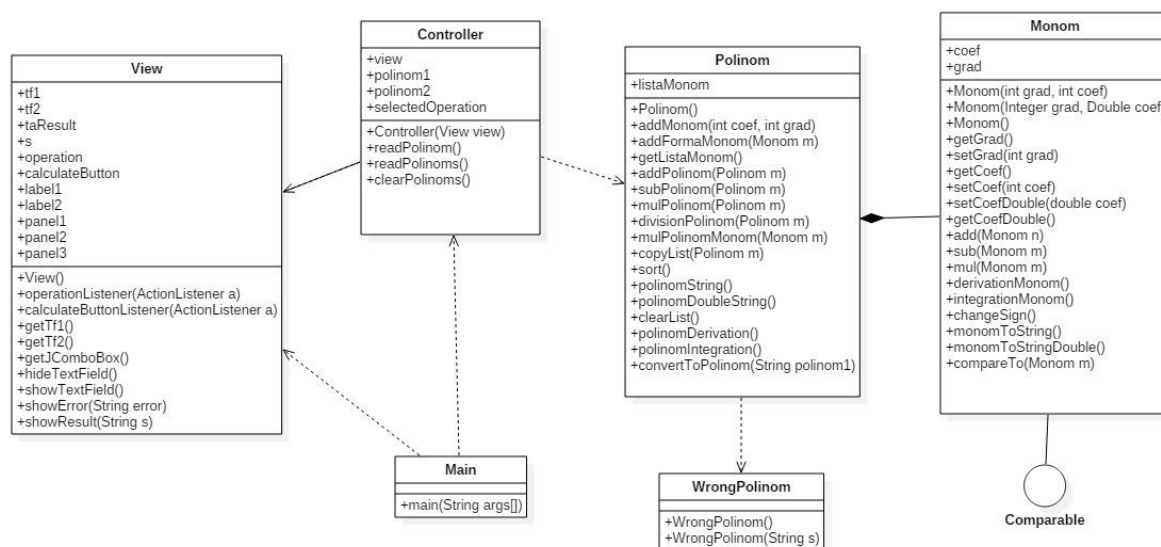
Putem încerca o abordare care să folosească memorii dinamice , o obținuie care poate fi luată în considerare ar fi utilizarea unor liste înlanțuite , care ar fi mult mai eficiente din punct de vedere al memorie oferindu-ne oportunitatea sa reține un numar nespecificat de monoame în fiecare polinom.

3.2. Clase

Funcționalitatea programului conține următoarele șapte clase. Clasa "main" reprezintă clasa care execută programul.

Celelalte clase "View", "Monom", "Polinom", "WrongPolinom", "Controller" contribuie la crearea designului MVC. Interfața grafică este implementată de clasa "view" care moșteneste clasa JFrame. În clasa "Controller" vom face legăturile dintre pechetele "view" și "model" pentru a sigura un răspuns vizibil utilizatorului. Celelalte clase "Monom" și "Polinom" calculează rezultatele efectuate pe polinoame, iar clasa "WrongPolinom" extinde clasa Exception și aruncă o excepție atunci când este nevoie (folosită doar în cazul în care polinomul introdus nu este corect).

Diagrama UML de clase



3.3 Algoritmi folosiți

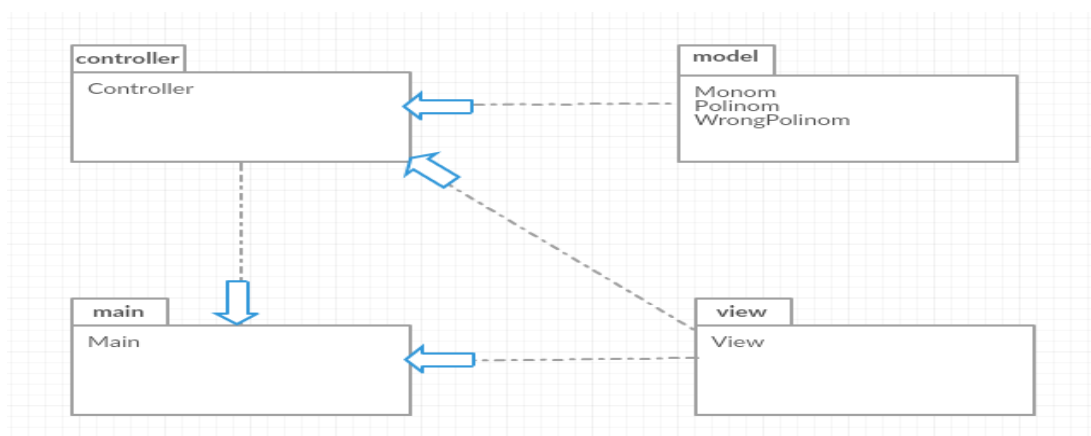
Algoritmi folosiți în descrierea și modelarea operațiilor cu polinoame sunt matematici, bazati pe formule matematice și metode de calcul. Prin urmare algoritmi pentru adunare și scădere sunt destul de simpli, conform matematicii doar trebuie să adunăm/ scădem coeficienți elementele care au același grad din. În cazul înmulțirii fiecare monom va fi al polinomului 1 va fi înmulțit cu fiecare monom al polinomului 2, rezultatele fiind adunate ulterior. Pentru împărțire am folosit algoritmul lui Euclid. Pe lângă algoritmi cu două polinoame trebuie implementați și algoritmi ce efectuează operații pe un singur polinom

adică derivarea și integrarea. Acești algoritmi au fost ușor implementați bazându-ne pe relațiile dintre grad, coeficient și formulele matematice.

Chiar dacă cel mai probabil există alți algoritmi mult mai eficienți pentru rezolvarea cerințelor, cei aleși oferă posibilitatea de a oferi șansa oricărei persoane cu cunoștințe medii să înțeleagă codul atunci când îl vede pentru prima dată.

3.4. Pachete

Cât posibil am încercat încadrarea fiecărei clase în câte un pachet, astfel am ajuns să folosim pachetele: model, view, controller, main. Pachetul model conține clasele Monom, Polinom și WrongPolinom (folosită pentru aruncarea unei excepții în cazul în care am introdus un polinom greșit). Celelalte pachete conțin câte o clasă fiecare denumite la fel ca pachetul din care face parte.

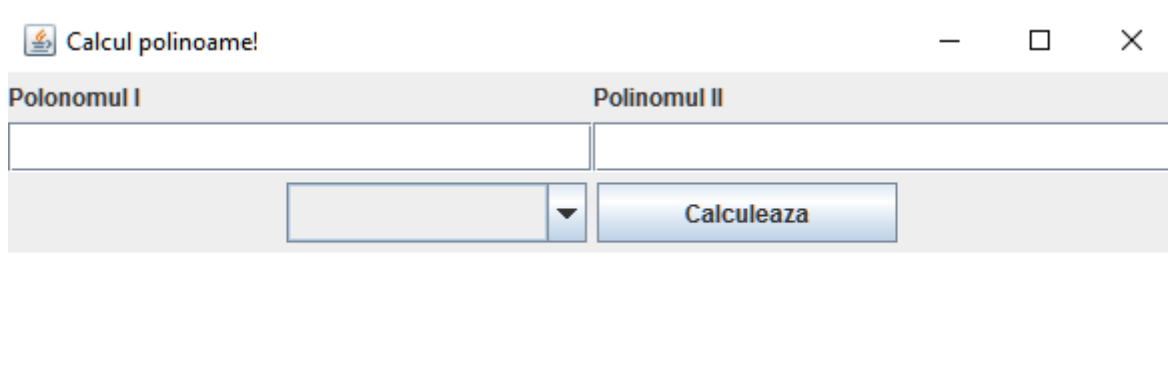


3.5. Interfețe

Clasa Monom implementează interfața Comparable pentru a facilita sortarea listei de monoame conținută de fiecare polinom în parte. Implementarea acestei interfețe permite suprascrierea metodei compareTo().

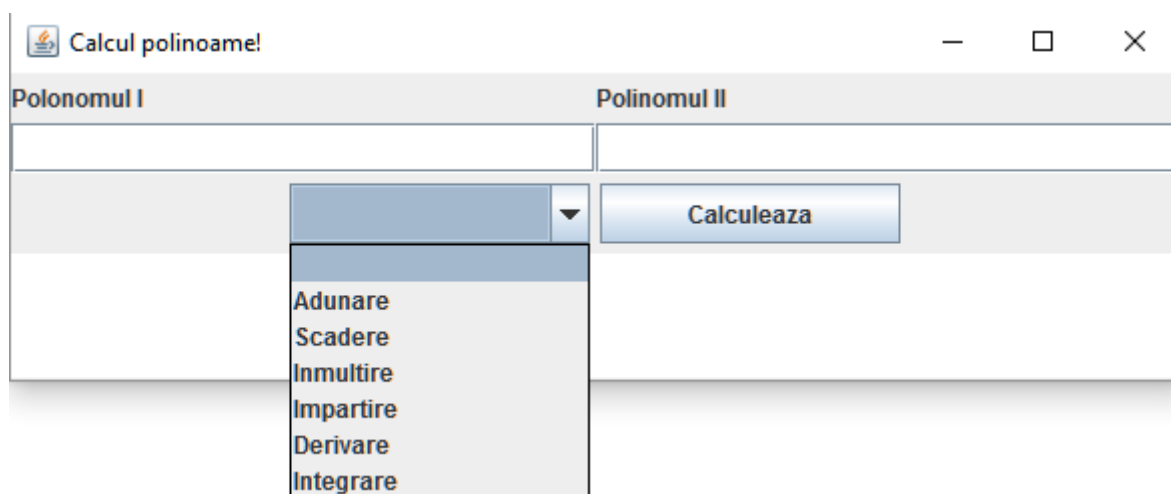
3.6. Interfața cu utilizatorul

Este folosită o interfață grafică foarte simplă și destul de intuitivă care conține două TextFielduri, două Labeluri, un TextArea, un ComboBox și un Buton.

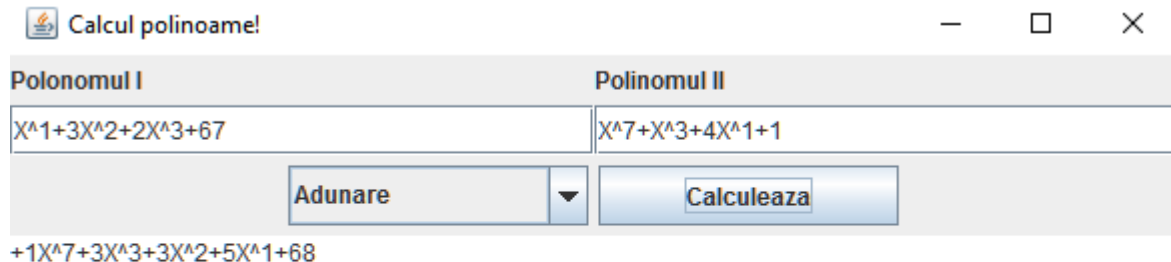


În acord cu regulile de scriere a polinomului, utilizatorul trebuie să introducă primul polinom în TextFieldul numit "Polinomul I", iar cel de-al doilea în celălalt camp numit "Polinomul II". Odată ce datele au fost introduse mai rămâne doar să selectăm operația pe care dorim să o efectuăm (operații prezente în ComboBox așa cum am menționat și înainte). După alegerea operației și apăsarea butonului "Calculeaza" rezultatul va apărea în câmpul TextArea aflat chiar sub ComboBox și Buton.

Am ales să folosesc acest ComboBox pentru a oferi utilizatorului o ușoară aplicație. Motivele alegerii câmpului TextArea au fost strict din cauza rezultatului care de multe ori nu poate fi reprezentat într-un simplu label.



În imaginea de mai sus putem observa cum apar operațiile în ComboBox.



Calcul polinoame!

Polinomul I: $X^1+3X^2+2X^3+67$

Polinomul II: $X^7+X^3+4X^1+1$

Adunare

Calculeaza

+1X⁷+3X³+3X²+5X¹+68

Aici putem observa cum aplicația știe să se ocupe de anumite evenimente și cum interfața cu utilizatorul știe să afișeze rezultatele.

4. Implementare și Testare

Programul a fost scris prin implementarea claselor deja menționate. În fiecare clasă au fost implementate metode folosind algoritmi deja descriși mai sus, acestea au fost declarate public, iar câmpurile private fiind accesibile doar cu ajutorul getterelor și setterelor pentru a asigura permisiunile la aceste date.

Clasa Monom:

```
package model;
import java.util.*;
public class Monom implements Comparable<Monom>{
    private Number grad,coef;

    //Constructori
    public Monom(int grad,int coef)
    {this.grad=grad;
    this.coef=coef;
    }
    public Monom(Integer grad,Double coef)
    {this.grad=grad;
    this.coef=coef;
    }
    public Monom(){
    this.grad=0;
    this.coef=0;
    }
}
```

Prima dată a fost specificat pachetul din care face parte clasa, librăriile care au fost importate, declarația clasei, câmpurile și constructorii. Așa cum se poate vedea clasa Monom

este implementată de interfața Comparable, care cum am explicat și mai sus se ocupa de sortare listei de monoame în ordine descrescătoare după grad.

Câmpurile declarate sunt sugestiv denumite astfel grad face referire la puterea monomului, iar coef la coeficientul său. Acestea au tipul Number care ne permite să reținem atât valori întregi cât și valori reale, din cauza faptului că este extins atât de tipul Double cât și de tipul Integer.

```
//Getters și Setters
public int getGrad() {
    return grad.intValue();
}
public void setGrad(int grad) {
    this.grad = new Integer(grad);
}
public int getCoef() {
    return coef.intValue();
}
public void setCoef(int coef) {
    this.coef = new Integer(coef);
}
public void setCoefDouble(double coef)
{ //Trunchiem coeficientul ca să nu aibă zerouri
    double result = Math.round(coef * 100) / 100.0;
    this.coef = result;
}
public double getCoefDouble()
{ return this.coef.doubleValue();
}
```

Metodele care ne oferă posibilitatea să setăm sau să primim valoarea câmpurilor clasei "Monom".

Pe lângă aceste metode mai avem și metodele care se ocupă cu operațiile pe monoame. Fiecare apel al unei metode care efectuează câte o operație va returna câte un obiect de tipul Monom, lucru care ușurează operațiile cu polinoame.

```
//Metoda pentru adunarea a doua monoame
public Monom add(Monom n)
{Monom result=new Monom();
    result.setCoef(this.coef.intValue()+n.getCoef());
    result.setGrad(this.grad.intValue());
return result;
}

//Metoda pentru scaderea a doua monoame
public Monom sub(Monom m)
{Monom result=new Monom();
    result.setGrad(this.grad.intValue());
    result.setCoef(this.coef.intValue()-m.getCoef());
return result;
}

//Metoda pentru inmultirea a doua monoame
public Monom mul(Monom m)
{
    return new Monom(this.grad.intValue()+m.getGrad(),this.coef.intValue()*m.getCoef());
}

//Metoda pentru derivarea unui monom
public Monom derivationMonom()
{Monom result=new Monom();
    result.setCoef(this.coef.intValue()*this.grad.intValue());
    result.setGrad(this.grad.intValue()-1);
return result;
}

//Metoda pentru integrarea unui monom
public Monom integrationMonom()
{Monom result =new Monom();
    int gradC=this.grad.intValue()+1;
    result.setGrad(gradC);
    result.setCoefDouble((double)this.coef.intValue()/gradC);
return result;
}
```

Metodele prezentate mai sus efectueaza operati aritmetice simple asupra unui monom în care se cunosc cele două câmpuri . Adicional am mai introdus încă o metodă care convertește un monon la un șir de caractere ,ajutând astfel la afisarea polinomului (numele metodei monomToString()).

Clasa Polinom:

Clasa Polinom face parte din pachetul "model" împreună cu clasele Monom și WrongPolinom, acesta are doar un camp care reprezintă lista de monoame denumita listaMonom. Am ales să prezint în continuare metodele mai interesante din punct de vedere al implementării lor.

În continuare voi spune câteva cuvinte despre metoda care primește ca parametru un String și-l convertește la tipul Polinom, creează o lista de monoame în funcție de datele introduse în TextField. Această implementare este destul de simplă, dar totuși destul de greu de implementat. Principiul de funcționare este următoarea :

1. verificăm dacă primul element este un semn ("+" sau "-")
2. aflăm coeficientul dacă există altfel vom pune valoarea 1, în cazul în care coeficientul este format din mai multe cifre vom crea un substring care mai apoi îl convertim la tipul int
3. examinăm următorul caracter să fie "X" sau "x"
4. după care revenim la pasul 1

În cazul în care unul dintre pași nu este satisfăcut vom arunca o excepție de tipul WrongPolinom.

Operațiile executate asupra polinoamelor sunt destul de simple în continuare am ales să prezint implementările pentru adunare, înmulțire, împărțire și integrare. Toate metodele care vor fi prezentate vor returna un obiect de tip polinom.

```
//Adunarea a doua polinoame
public Polinom addPolinom(Polinom m)
{Polinom resultAdd=new Polinom();
 int nrM=m.getListaMonom().size();
 int a[]=new int[nrM];
 for(Monom monom:this.listaMonom)
 {int grad=monom.getGrad();
  int i=0;
  boolean check=true;
  for(Monom monom1:m.getListaMonom())
  {if(grad==monom1.getGrad())
  {resultAdd.addFormaMonom(monom.add(monom1));
   a[i]=1;
   check=false;
   i++;
  }
  else i++;
  }
  if(check)resultAdd.addFormaMonom(monom);
 }
 int i=0;
 for(Monom monom1:m.getListaMonom())
 {if(a[i]==0)resultAdd.addFormaMonom(monom1);
  i++;
 }
 return resultAdd;
}
```

Metoda care implementează adunarea a doua polinoame adunam fiecare monom din polinom cu care am apelat metoda la fiecare monom din polinomul m primit ca argument (suma a doua monoame se face prin simplul apel al metodei add definită în clasa Monom), în plus necesită verificarea dacă nu cumva mai trebuie adăugate monoame din m (există posibilitatea ca polinomul m să aibă termeni care nu au gradul egal cu niciun monom din polinomul this).

```
//Înmulțirea a doua polinoamelor
public Polinom mulPolinom(Polinom m)
{Polinom result=new Polinom();
for(Monom monom:this.listaMonom)
    for(Monom monom1:m.getListaMonom())
    {    boolean check=true;
        Monom resultMonom=monom.mul(monom1);
        int gradResult=resultMonom.getGrad();
        for(Monom checker:result.getListaMonom())
            if(checker.getGrad()==gradResult)
            {result.addFormaMonom(checker.add(resultMonom));
             checker.setCoef(0);
             checker.setGrad(0);
             check=false;
             break;
            }
        if(check)result.addFormaMonom(resultMonom);
    }
result.sort();
return result;
}
```

Înmulțirea polinoamelor este aproximativ identică ca implementare cu adunarea singura diferență constă în apel funcției mul pentru o pereche de monoame și verificare existenței unui monom cu același grad în polinomul result. În cazul în care găsim două monoame cu grade egale vom apela funcția add (folosită și la operația de adunare).

```
//Impartirea a doua polinoame
public String divisionPolinom(Polinom m)
{Polinom quotient=new Polinom();
 Monom firstElement=listaMonom.get(0);
 this.sort();
 int grad=firstElement.getGrad();
 int gradDivisor=m.getListaMonom().get(0).getGrad();
 double coefDivisor=m.getListaMonom().get(0).getCoef();
 m.sort();
 if(grad<gradDivisor)return "Restul este:"+this.polinomString();
 else {
 while(grad>=gradDivisor)
 {
 //Determinam gradul si coeficientul catului ,cream un nou monom si-l adaug:
 int gradCat=grad-gradDivisor;
 double coefCat=coefDivisor*firstElement.getCoef()/coefDivisor;
 Monom quotientMonom=new Monom(gradCat,coefCat);
 quotient.addFormaMonom(quotientMonom);

 //Efectuam operatiile de scadere,inmultire
 Polinom reminder=new Polinom();
 reminder=m.mulPolinomMonom(quotientMonom);

 Polinom acc=this.subPolinom(reminder);//Retinem polinomul care se obtine di

 this.copyList(acc);//Copiem continutul in lista

 //Verificam daca lista mai contine elemente
 if(!this.listaMonom.isEmpty())firstElement=listaMonom.get(0);
 else break;
 grad=firstElement.getGrad();
 }
 return"Catul :"+quotient.polinomString()+" Restul : "+this.polinomString();
 }
 }
```

Cea mai interesanta operație este împărțirea pentru implementarea acesteia am folosit algoritmul lui Euclid ,care folosește operații clasice de adunare, scădere și înmulțire cu un coeficient. Aceste operații ajutătoare se vor repeat atâta timp cât gradul monomului cel mai mare din primul polinom (this) este mai mare sau egal cu gradul monomului din cel de-al doilea polinom.

```
//Determinare polinomului integrat
public Polinom polinomIntegration()
{Polinom result=new Polinom();
 for(Monom monom:this.listaMonom)
 result.addFormaMonom(monom.integrationMonom());
 return result;
 }
```

Integrarea unui polinom nu are nevoie de niciun parametru, vom lucra chiar pe polinomul cu care am apelat metoda. În interiorul acestei metode se face un apel la funcția integrationMonom care returnează un monom cu coeficientul de tip Double. Pentru a afișa polinomul rezultat am făcut o metodă special numită polinomDoubleString() care returnează un String.

Clasa View:

Cea mai importantă parte din view este constructorul care conține informații despre crearea interfeței grafice. Unele metode apelate în interiorul acestui constructor sunt doar câteva metode auxiliare create în interiorul acestei clase cu scopul de a crește înțelegerea codului.

```
//Constructorul
public View()
{
    //Formare fereastra
    this.setTitle("Calcul polinoame!");
    this.setVisible(true);
    this.setSize(600,190);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Setare design și componente
    taResult.setEditable(false);
    taResult.setPreferredSize(new Dimension(20,20));

    //Primul panou
    label1.setText("Polinomul I");
    label2.setText("Polinomul II");
    tf1.setPreferredSize(new Dimension(30,30));
    tf2.setPreferredSize(new Dimension(30,30));
    panel1.setLayout(new GridLayout(2,2));
    panel1.setPreferredSize(new Dimension(50,50));
    panel1.add(label1);
    panel1.add(label2);
    panel1.add(tf1);
    panel1.add(tf2);

    //Al doilea panou
    panel2.setLayout(new FlowLayout());
    panel2.add(operation);
    panel2.add(calculateButton);
    operation.setSelectedIndex(0);
    operation.setPreferredSize(new Dimension(150,30));
    calculateButton.setPreferredSize(new Dimension(150,30));

    //Al treilea panou
    panel3.setLayout(new BoxLayout(panel3,BoxLayout.Y_AXIS));
    panel3.add(panel1);
    panel3.add(panel2);
    panel3.add(taResult);

    this.setContentPane(panel3);
}
```

Ascultătorii pentru butoan și combobox au fost declarați în controller, iar ulterior au fost apelate metode care să asocieze fiecărui element câte un ascultător. De exemplu un ascultător pentru combobox este :

```
//Listenerele pentru butoane și combobox
public class changeList implements ActionListener{
    public void actionPerformed(ActionEvent a)
    {
        selectedOperation=view.getJComboBox();
    }
}
```

JUnit Tests:

O serie de teste au fost facute pentru a asigura o bună comportare a acestei aplicații. În cadrul clasei TestPolinom au fost declarate și instanțiate două polinoame. Mai apoi după testele sunt făcute ,iar rezultatele metodei assert sunt vizibile, dacă afirmația nu este adevărată testul va fi considerat eșuat altfel va fi considerat un success. După terminarea unui test se reiau inițializare și toate celelalte teste.

```
@Before
public void setUp()
{
    try{p1=new Polinom();
    p1.convertToPolinom("X^1+3X^2+2X^3+67");
    p2=new Polinom();
    p2.convertToPolinom("X^7+X^3+4X^1+1");
    }
    catch(WrongPolinom e)
    {
        assertNotNull(e);
    }
}
```

Runs: 6/6 Errors: 0 Failures: 1

model.TestPolinom [Runner: JUnit 5]


- testPolinomIntegration (0,000 s)
- testPolinomDivision (0,000 s)
- testAddPolinom (0,000 s)
- testPolinomMul (0,000 s)
- testPolinomSub (0,004 s)
- testPolinomDerivation (0,004 s)

Failure Trace

```
org.junit.ComparisonFailure: expected:<[
at model.TestPolinom.testPolinomSub(T
at java.util.stream.ForEachOps$ForEachC
at java.util.stream.ReferencePipeline$3$1
at java.util.Iterator.forEachRemaining(Ur
at java.util.Spliterators$IteratorSpliterato
at java.util.stream.AbstractPipeline.copy
at java.util.stream.AbstractPipeline.wrap
at java.util.stream.ForEachOps$ForEachC
at java.util.stream.ForEachOps$ForEachC
at java.util.stream.ReferencePipeline.forf
```

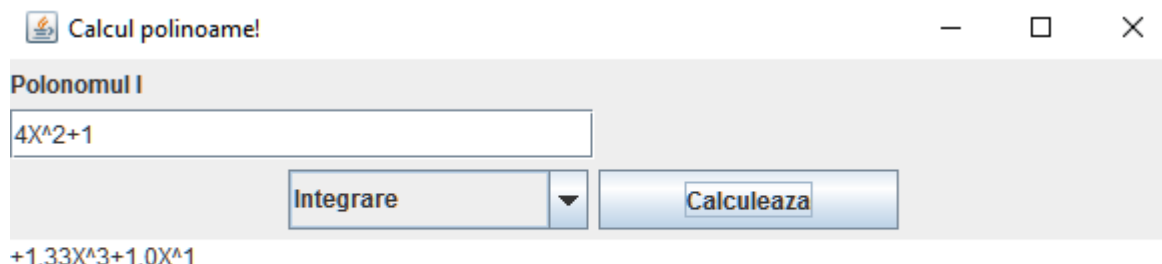
5.Rezultat

Rezultatul este un calcul aritmetic pe două polinoame, fie o operație aplicată asupra unui polinom. Acesta depinde doar de datele de intrare si de operații pe care dorim să le efectuăm.


Calcul polinoame!
— □ ×

Polinomul I	Polinomul II
X^1+1	X^3+1
<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid #ccc; padding: 5px 10px;">Adunare ▼</div> <div style="border: 1px solid #ccc; padding: 5px 10px;">Calculeaza</div> </div>	
+1X^3+1X^1+2	

Pentru datele de intrare introduce în câmpuri vom obține rezultatul asteptat pentru operația de adunare.



Dacă alegem operația de integrare vom putea observa rezultatul care conține coeficienți reali.

6. Concluzie

Chiar dacă această aplicație este un simplu calculator de polinoame de viitor am putea încerca realizarea mai operațiilor cu mai multe polinoame, adăugarea unui graf care să reprezinte funcția de ieșire, algoritmi mult mai eficienți din punct de vedere al complexității și memoriei ocupate.

7. Bibliografie

- <http://users.utcluj.ro/~igiosan/>
- <https://www.youtube.com/watch?v=l8XXfgF9GSc>
- <https://www.youtube.com/watch?v=VZpXQCDMYHQ>
- <https://stackoverflow.com/questions/11229986/get-string-character-by-index-java>
- <https://stackoverflow.com/questions/21626439/how-to-implement-the-java-comparable-interface>