**Documentatie**

la disciplina

**Tehnici de programare**

**Tema 1 – Calulator de polinoame**

**Popa Cristian Mihai, grupa 30224**

**An academic: 2021-2022**

Cuprins

[1. Obiectivul temei: 3](#_Toc97863049)

[2. Analiza problemei: 3](#_Toc97863050)

[3. Proiectare: 4](#_Toc97863051)

[4. Implementare: 6](#_Toc97863052)

# Obiectivul temei:

* **Obiectiv principal:**

Obiectivul acestei teme consta in creare unei aplicații utila, eficienta si ușor de utilizat in vederea procesării calculelor ce se pot aplica unui polinom. Un polinom este compus din mai multe monoame. Monomul este constituit dintr-un coeficient reali si o putere(in proiect denumita exponent) reala, care se atașează variabilei necunoscute X.

Aplicația trebuie sa primească input de la utilizator, sub forma de un string ce reprezintă un polinom, iar cu ajutorul butoanelor ce se afla pe interfața, utilizatorul va putea vizualiza rezultatul operației selectate.

* **Obiectiv secundare**:
  + Revizualizarea unor paradigme de programare orientata pe obiect: Obiect, Clasa, Interfețe, Moștenire etc.;
  + Dezvoltarea unor clase de obiect adecvate: crearea unor clase potrivite pentru monom si polinom, astfel încât sa informația sa fie ușor de sintetizat;
  + Dezvoltarea unor algoritmi eficienți si ușor de înțeles pentru prelucrarea polinoamelor;
  + Implementarea operațiilor specifice pentru polinoame, si cum pot fi simplificate;
  + Implementarea unei interfețe grafice ușor de utilizat si ușor de înțeles;
  + Implementarea soluției pentru a avea un calculator funcțional de polinoame

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

# Analiza problemei:

Calculele cu polinoame, exceptând adunarea si scăderea(deși pentru polinoame foarte lungi reprezintă si ele o povara), sunt foarte dificile si necesita un timp foarte lung pentru rezolvarea si verificarea rezultatului. Deși avem algoritmi si pentru cele mai dificile operații, cum ar fi împărțirea, calculatorul facilitează acești pași. Utilizatorul va trebui sa introducă doar cei doi polinom in cele doua TextField-uri de pe interfața, iar după alegere, va apăsa un buton din cele 6 pentru alegerea operației dorite. Cele 6 operați de care dispune programul sunt:

* Adunare;
* Scădere;
* Înmulțire;
* Împărțire;
* Derivare;
* Integrare;

In cazul celor doua operații de derivare si integrare, doar polinomul din primul TextField va fi afectat. In rest, rezultatul va fi dat in urma prelucrării celor doua polinoame, in funcție de operația selectata.

Pentru a funcționa corect, programul presupune introducerea polinoamelor sub o anumita forma, ușor de înțeles. De exemplu, pentru un polinom de gradul 3 vom avea: x3−12\*x2+38x−17 devine x^3 – 12x^2+38^x-17. Semnul de înmulțire „\*” dispare iar ridicarea la putere se înlocuiește cu „^”, urmat de exponentul dorit. De asemenea, daca se dorește introducerea unor termeni liberi, ei se pot scrie normal, fara a fi necesar termenul „x^0”.

Programul verifica fiecare input al utilizatorului, deoarece pot apărea greșeli(de exemplu: utilizatorul introduce alta variabila in loc de x sau poate pune X mare) si încerca sa corecteze pe cat posibil problema. In cazul unei erori, utilizatorul va fi notificat, iar după poate sa introducă alt polinom, programul va rula in continuare. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

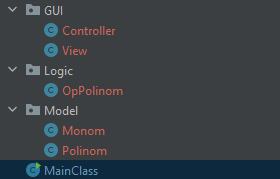
După introducerea celor doua polinoame si selectarea operației dorite, rezultatul va apărea in cel de-al 3-lea TextField. De asemenea, exista un buton numit „Exemplu” care poate fi folosit pentru introducerea rapida a doua polinoame de grad diferit pentru a se vedea cum funcționează fiecare operație, fără ca utilizatorul sa mai fie nevoit sa introducă pentru început doua polinoame. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

# Proiectare:

* **Alegerea unor clase de obiecte de baza adecvate:**
  + Un polinom poate fi format dintru-unul sau mai multe monoame. Un monom reprezintă o singura variabila x, ce poate fi însoțita de un coeficient si un exponent. De asemenea, si termenii liberi pot fi priviți ca un monom, de exemplu fie un număr W, el poate fi scris ca W\*x^0.
  + Monomul este descris de clasa Monom, ce poseda următoarele atribute: „nr” sau coeficient(cel din fata variabilei x) ce este de tip real si „exp” ce reprezintă exponentul unui monom si este de tip întreg. La început se poate opta pentru introducerea de coeficienți reali, dar adevăratul motiv pentru care „nr” este real este pentru a putea fi realizate operațiile de împărțire, derivare si integrare mai ales.
  + Monomul poseda câteva metode, getter si setter, doi constructori(unul pt a construi un monom cu coef. si exp. dat si altul pentru a copia atributele altui monom) si alte metode de afișare si o metoda de comparație.
  + Clasa Polinom poseda un singur atribut privat, ce consta într-o lista (Array List) de Monoame. Deși nu are constructori speciali, poseda mai multe metode decât fratele mai mic, Monom. Aceste metode ne ajuta in formarea de noi polinoame, in aranjarea lor si facilitează realizarea anumitor operații.
  + Aceste doua clase stau la bazaz programului, iar ceea ce conține fiecare este foarte intuitiv si ușor de înțeles.
* **Împărțirea programului pe clase:**

Programul este împărțit in 3 pachete mari după modelul Model-View-Controller (MVC). Cele 3 pachete sunt:

* Pachetul GUI (Grafical User Interface) ce cuprinde 2 clase, Controller si View ce stau la baza funcționarii aplicației. Clasa View este responsabila pentru crearea interfeței grafice, iar clasa Controller ne ajuta sa interacționăm cu Text-filed-urile si butoanele de pe Form;
* Pachetul Logic conține toate cele 6 operații ce se pot aplica pe un polinom, astfel nefiind nevoiți sa adăugam operațiile ca metode in clasa Polinom;
* Pachetul Model conține clasele descrise mai sus: Monom si Polinom, cele ce ne descriu un polinom si atributele sale;
* In plus, mai avem o clasa numita „MainClass” ce nu aparține niciunei clase(deși s-ar putea încadra in clasa Model) care deschide interfața si inițializează controller-ul.
* Astfel, clasele sunt organizate in 3 categorii respectând principiul MVC.
* Împărțirea lor pe clase:

 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

* **Algoritmi folosiți:**

Cum prelucrează programul un string dat ca si input? Ei bine, sunt folosite expresii de pattern-matching RegeX, astfel fiecare polinom fiind împărțit in monoame(dar tot un sub forma de string), care la rândul lui este împărțit in coeficient si exponent. Programul poate identifica si trata si cazurile speciale: când avem ca prim termen un monom ce are coeficient pe 1(sau -1), si nu avem niciun semn in fata variabilei x(in cazul -1, avem semnul – simplu in fata lui x). Programul identifica eficient si clar coeficienții si exponenții variabilelor necunoscute x.

Pentru a evita anumite probleme, in momentul in care utilizatorul introduce un polinom, dar gradele nu sunt într-o ordine, programul ordonează monoamele in mod descrescător in funcție de monomul cu gradul cel mai mare, fără a fi necesate intervenția utilizatorului si fără a schimba coeficienți si exponenții. De asemenea, aceasta reordonare are ca scop si eliminarea monoamelor cu exponent asemănător, adică formarea unui singur monom, prin adunarea duplicatelor.

Adunarea se face pe baza acestui algoritm de reordonare. Pur si simplu cream un string care reprezintă un polinom format prin alipirea celor 2 polinoame si semnul „+” intre ele. Programul convertește string-ul respectiv, iar in urma conversiei, vom avea rezultatul celor 2 polinoame adunate.

Analog se face si scăderea, dar înainte se înmulțesc toți coeficienți celui de-al doilea polinom cu -1, se formează noul String pentru cel de-al doilea polinom, iar după se face adunare, ca mai sus, intre cele doua polinoame.

Operațiile de înmulțire, derivare si integrare se fac după cum suntem obișnuiți, după regulile matematice.

Cel mai complex algoritm de prelucrare a polinoamelor este cel de împărțire, care se realizează conform următorilor pași:

* Se ordonează descrescător după exponent monoamele din cele doua polinoame;
* Se împarte polinomul cu monomul cu gradul cel mai ridicat la celălalt polinom;
* Se împarte primul monom din primul polinom cu primul monom din al doilea polinom si se obține primul termen din cat;
* Termenul obținut trebuie înmulțit cu împărțitorul astfel încât sa obținem un polinom care elimina cel mai mare monom din deîmpărțitor;
* Noul polinom obținut este scăzut din deîmpărțitor, si se obține un rest;
* Atât timp cat restul are un monom cu grad mai mare decât împărțitorul, se repeta pașii de mai sus, dar noul deîmpărțitor devine restul obținut anterior; . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
* La final vom avea un rest si un cat, verificarea se poate face daca înmulțim catul cu împărțitorul si adunam restul;

# Implementare:

In cele ce urmează, se vor prezenta modul in care au fost implementate clasele de obiecte, din cele 3 pachete. Nu vor fi incluse setter si getter, deoarece se subînțelege ca au fost folosite.

* **Pachetul Gui:**
  + Clasa View:

package GUI;

public class View extends JFrame{....}

Clasa View conține si inițializează toate componentele vizuale menționate mai sus: 7 butoane, 3 TextField-uri si 3 laber-uri. Este sunt plasate un panouri invizibile care la rândul lor aranjate intr-un mod specific in fereastra principala. Câteva inițializări si metode din clasa View:

private JPanel panel7 = new JPanel();  
private Font font1 = new Font("SansSerif", Font.*BOLD*, 15);  
  
private JLabel label1 = new JLabel("Primul polinom:");  
private JLabel label2 = new JLabel("Al doilea polinom:");  
private JLabel label3 = new JLabel("Rezultat");  
  
private JTextField text1 = new JTextField(40);

public JTextField getResult() {  
 return text3;  
}  
  
public void ClearResult(){  
 this.text3.setText("");  
}

* + Clasa Controller:

public class Controller implements ActionListener{  
 private View view;  
  
 private OpPolinom op = new OpPolinom();  
  
 public Controller(View v){  
 this.view = v;  
 }

....

}

Poate fi considerat creierul aplicației, întrucât comunica intre interfața vizuala si logica din spatele programului, ajuta la rezolvarea operației selectate si la livrarea rezultatului. Câteva metode, ca exemplu:

public void actionPerformed(ActionEvent e) {  
 String command = e.getActionCommand();  
 String s1 = view.getFirstPolinom().getText();  
 String s2 = view.getSecondPolinom().getText();  
 int ok=1;  
  
 Polinom p1 = new Polinom();p1.fString(s1);  
 Polinom p2 = new Polinom();p2.fString(s2);  
 Polinom rez = new Polinom();  
  
 switch (command)  
 {  
 case"Adunare":  
 rez=op.Add(p1,p2);  
 break;  
 case"Scadere":  
 rez=op.Sub(p1,p2);  
 break;

...}

* **Pachetul Logic:**
  + Clasa OpPolinom:

public class OpPolinom {. . .}

In aceasta clasa sunt descrise operațiile ce se pot aplica pe polinoame, cum ar fi:

public Polinom Add(Polinom p1, Polinom p2)  
{  
 Polinom rez = new Polinom();  
 rez.fString(p1.getPolString() + "+" + p2.getPolString());  
 return rez;  
}  
  
public Polinom Sub(Polinom p1, Polinom p2)  
{  
 Polinom rez = new Polinom();  
 p2.mulScalar(-1);  
 rez.fString(p1.getPolString() + "+" + p2.getPolString());  
 return rez;  
}

public Polinom Ing(Polinom p)// integrare  
{  
 float coef=0;  
 int exp=0;  
 Polinom rez = new Polinom();  
  
 for(Monom m: p.getPolinom())  
 {  
 coef=m.getNr();  
 exp=m.getExp()+1;  
 rez.add(new Monom(coef/exp, exp));  
 }  
 return rez;  
}

* **Pachetul Model:**
  + Clasa Monom: este clasa ce sta la baza aplicație

public class Monom implements Comparable<Monom>{  
 private float nr;  
 private int exp;  
  
 public Monom(){};  
  
 public Monom(float a, int b){  
 this.nr=a;  
 this.exp=b;  
 }  
  
 public Monom(Monom m){  
 this.nr=m.getNr();  
 this.exp=m.getExp();  
 }

. . . }

Se pot observa cei doi constructori, după cum au fost descriși mai sus.

Dintre metodele importante, regăsim:

public int compareTo(Monom o) {  
 if(this.exp >= o.exp)  
 return -1;  
 return 1;  
}

Ne ajuta pentru sortarea automată din biblioteca Collections. Restul metodelor sunt folosite pentru afișare, returnează un String sub forma „2x^3”, de exemplu.

* + Clasa Polinom:

public class Polinom {  
 private List<Monom> polinom = new ArrayList<Monom>();

. . .}

Aceasta clasa posedeaza un singur atribut, ce este un Array List de monoame. In aceasta clasa regăsim tot felul de metode folositoare pentru prelucrarea polinoamelor:

public void reorder()  
{  
 Collections.*sort*(this.polinom);  
  
 for(int i=0; i<polinom.size(); i++)  
 {  
 for(int j=i+1; j<polinom.size(); j++)  
 {  
 if(polinom.get(i).getExp() == polinom.get(j).getExp())  
 {  
 polinom.get(i).setNr(polinom.get(i).getNr() + polinom.get(j).getNr());  
 polinom.remove(j);  
 j--;  
 }  
 }  
 }  
}

Aceasta metoda ne ajuta sa rearanjam monomii, sa-i adunam pe cei care au exponenții asemănători. Aceasta este metoda ce sta la baza adunării dintre cele 2 polinoame.

public void mulScalar(int x)  
{  
 for(Monom m: polinom)  
 {  
 m.setNr(m.getNr() \* x);  
 }  
}  
  
public void mulMonom(Monom m)  
{  
 for(Monom myMonom: polinom)  
 {  
 myMonom.setNr(myMonom.getNr()\*m.getNr());  
 myMonom.setExp(myMonom.getExp()+m.getExp());  
 }  
}

Cele doua metode de mai sus oferă posibilitatea înmulțirii polinomului nostru cu un scalar sau cu un monom, util pentru împărțire sau înmulțire.

public void getRidOfZero()  
{  
 for(int i=0; i<polinom.size(); i++)  
 {  
 if(polinom.get(i).getNr()==0)  
 polinom.remove(i);  
 }  
}

Metoda este utila in momentul in care facem împărțirea, vom avea polinoame care in urma împărțirii vor conține monoame ce au coeficient zero si exponent zero, echivalentul numărului 0, ceea ce nu avem nevoie.

Unul dintre aspectele importante de menționat este prelucrarea string-ului ce reprezintă un polinom. Acest lucru se realizează cu ajutorul expresiilor RegeX, expresii folosite pentru patter-matching. In cazul meu, expresia folosita de mine a fost:

String polinom\_pattern = "[+-]?[^-+]+"

In rest, clasa polinom conține câteva metode ce returnează polinomul sub forma de string, sub diferite formate( in funcție de ce se dorește, prelucrare sau afișare).

1. **Concluzie:**

Aplicația dezvoltata respecta paradigmele programării orientate pe obiect si este un bun reminder al cunoștințelor învățate sem. trecut. Calculatorul funcționează in conformitate cu cerința daca utilizatorul respecta convenția de introducere a polinoamelor, deși programul poate repara anumite erori. Programul a fost testat de mine, manual, luând tot felul de polinoame si cazuri speciale(un polinom si o constanta, polinoame cu exponent mare etc.)

Personal, acest proiect a fost o plăcere pentru mine, deoarece încă din liceu am avut o pasiune pentru limbajele POO(cu C# m-am învățat) si cream o grămadă de aplicații vizuale. In cazul acestui proiect m-am familiarizat cu expresiile de pattern-matching, ce ușurează foarte mult viața unui programator când vine vorba de prelucrarea precisa a unui String.

Deși este simpla aplicația si își deservește scopul, totuși este loc de îmbunătățiri si upgrade-uri. Aplicația poate avea pe viitor o interfața grafica mai bogata, ce sa permită mai multe operații ce se pot aplica pe un polinom, cum ar fi aflarea rădăcinilor, înmulțirea cu alt monom(deși operația este implementata, ea nu poate fi accesata de către utilizator) etc.

1. **Bibliografie:**

<https://www.emathhelp.net/en/calculators/algebra-1/polynomial-long-division-calculator/>

<https://docs.oracle.com/en/>

<https://www.baeldung.com/>

<https://www.jetbrains.com/help/idea/working-with-code-documentation.html>