

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**CATEDRA CALCULATOARE**

**DOCUMENTATIE TEMA 1**

**CALCULATOR POLINOAME**

**Tudor Colceriu**

**Grupa 30227**

**Profesor Laborator: Dan Mitrea**

**CUPRINS**

1. **Cerinte functionale**
2. **Obiective**
3. **Analiza proiectului**
4. **Proiectare + etape**
5. **Implementare + descriere de clase**
6. **Testare**
7. **Concluzii**
8. **Bibliografie**

# Cerinte Functionale

--- ------- --- -------- ------ ----------- ------- ----- ----- --------- ----- ---------- ------ ------- -- ------------- ---------------

Propuneti, proiectati si implementati un sistem de procesare a polinoamelor de o singura variabila cu coeficienti intregi.

Operatiile pe care le suporta programul sunt:

* Citirea unui polinom de la tastatura sub forma de String cu formatul a0x^n+a1x^(n-1)+a2x^(n-2)+...+a0, unde si
* Adunarea a doua polinoame P1(x) + P2(x)
* Scaderea a doua polinoame P1(x) - P2(x)
* Inmultirea a doua plinoame P1(x) \* P2(x)
* Impartirea a doua polinoame P1(x) / P2(x)
* Derivarea unui polinom P1(x)’
* Integrarea unui polinom ∫ P1(x)

# Obiective

## Obiectiv Principal:

Obiectivul principal al proiectului este de a creea o aplicatie care sa implementeze un sistem de procesare a polinoamelor ( acestea fiind construite din termeni numiti monoame, care sunt alcatuite dintr-o constanta (numita coeficient), inmultita cu una sau mai multe variabile; fiecare variabila poate avea un exponent constant intreg pozitiv).

## Obiective Secundare:

Dezvoltarea de use case-uri si scenarii: intr-un sistem software un use case reprezinta o lista de actiuni sau pasi de eveniment care defines in mod obisnuit interactiuniile dintre un rol (in limbaj UML – actor) si atingerea obiectivului propus. In cazul de fata, rolul il reprezinta utilizatorul, iar scenariul, interactiunea dintre acesta si programul creat.

Alegerea structurilor de date potrivite: structurile de date folosite pentru a duce la capat proiectul

Impartirea pe clase: utilizarea uui MVC (model view controller) pentru a forma o interfata grafica (GUI) prin care utilizatorul sa interactioneze cu programul scris.

Descrierea procedurilor: vor fi descrisi algoritmii ce au fost utilizati in realizarea proiectului, cat si structurile de date cu ajutorul carora opereaza. Mai mult de atat, clasele utilizate vor fi implementate, functionalitatile lor fiind specificate in particular, cat si deciziile care au dus la alegerea implementarii in acest mod. (de ce s-au folosit anumite clase) – acest lucru este legat de implementarea solutiei propuse

Testare: proiectul va contine cateva scenario de testare a operatiilor pe polinoame, utilizand ca instrument principal Junit Test.

# Analiza problemei

## Use case-uri si scenarii:

Utilizarea programului aflat la dispozitie consta strict in interactiunea utilizatorului cu interfata grafica (GUI) realizata. In momentul rularii, GUI-ul va fi vizibil si functional. Folosirea sa este foarte intuitiva, ea fiind simpla si concisa. In cele doua zone de text, marcate corespunzator, se vor introduce cele doua polinoame pe care se doreste a se realiza diferite operatii. Dupa introducerea de la tastatura a polinoamelor (respectand formatul impus de cerinta proiectului – vezi 1.Cerinte Functionale), operatiile vor fi realizabile cu ajutorul butoanelor aflate la dispozitie. Acestea contin un text intuitiv ce denota operatia ce va fi efectuata la apasarea butonului. Campurile 3 si 4 de text sunt utilizate pentru rezultate. Fara a rula sau a modifica nimic in interfata grafica, aceste campuri contin mesaje legate de modul de introducere si de afisare a rezultatului, astfel incat programul sa ruleze in conditii optime, iar operatiile sa produca rezultatul asteptat de catre utilizator.

Totodata, mesajele intuitive nefiind o bariera indeajuns de puternica impotriva erorii umane, String-uri sau polinoame eronate ar putea fi mereu introduse de la tastatura de catre utilizatori. In momentul introducerii unui mesaj ce nu respecta structura bine-definita de polinom, campul (JTextField) nu se va modifica !! Astfel se va semnala o greseala la datele de intrare oferite.

Programul este proiectat de asa natura incat, daca utilizatorul introduce de la tastatura in zonele de text pentru intrate (JTextField) un polinom care nu este ordonat intr-o maniera naturala (cu gradele in ordine crescatoare sau descrescatoare), programul va interpreta acest input ca fiind corect, rearanjand polinomul (in memoria sa) intr-un mod mai usor pentru realizarea operatiilor dorite mai departe.

## Diagrama de use case-uri:

Operatiile Implementate pe Polinoame:

Suma

Scaderea

Inmultirea

Impartirea

Derivarea

Integrarea

Rezultat

Afisat pe

GUI

# Proiectare (etape de proiectare):

## Structuri de date:

Structurile principale utilizate in proiectare sunt ArrayList-urile, ele avand o eficienta si o reprezentare mai buna, comparativ cu vectorii obisnuiti.

Din matematica se cunoaste faptul ca un polinom reprezinta o “colectie” de monoame, fiecare avand coeficienti si grade cu valori si semne diferite. Asadar, reprezentarea unui polinom, membru al clasei Polinom, este realizata utilizand o colectie de monoame (din clasa Monom) , prin structura ArrayList, formata din obiecte de tip Monom.

Sintaxa utilizata: List<Monom> polinom = new ArrayList<Monom>();

Asa cum s-a prezentat anterior, pentru definirea si utilizarea monoamelor se utilizeaza o clasa separata numita Monom. Ca si caracteristici, fiecare obiect, instanta a acestei clase, va avea 2 variabile predefinite: un coeficient (de tip double, asadar poate fi pozitiv sau negative, cu sau fara zecimale, tinandu-se cont de acest aspect) si o putere (un numar intreg). Coeficientul este de tip real, deoarece la anumite operatii (integrare sau diviziune de polinoame) se realizeaza impartirea coeficientilor unor polinoame cu anumite constante, rezultand numere reale, ce trebuie afisate cu zecimale dupa virgula, pentru o mai buna reprezentare si interpretare a rezultatului.

## C:\Users\Tudor\IdeaProjects\calcPolin\src\diagram.png

## 4.2 Diagrama de clase (UML):

# Implementare:

## Clasa Monom:

Clasa monom este practic cea care sta la baza unui polinom. Aceasta este implementata utilizand 2 campuri de tipul private, specifice fiecarui monom:

private double coeficient;  
private int putere;

Fiecare monom are in componenta sa un coeficient, numar real (datorita impartirilor ce sunt efectuate) si o putere, numar intreg. Fiind campuri de tip private, metode de get si set au fost generate pentru fiecare din ele.

Avand in vedere aceste doua tipuri, constructorul pentru un obiect monom primeste 2 argumente, un numar real si un numar intreg, corespondente pentru coeficient si putere.

## Clasa Polinom:

Clasa polinom face practic majoritatea operatiilor din proiect, folosind ca si sprijin clasa prezentata anterior, Monom (deoarece in reprezentarea matematica, un polinom este alcatuit din monoame).

Avand in vedere acest lucru, clasa polinom va avea in componenta fiecarui obiect creat o lista de monoame, definita astfel:

private List<Monom> termeni = new ArrayList<Monom>();

Deoarece in momentul in care se creaza un nou polinom nu dorim initializarea lui imediata, ci termenii vor fi adaugati pas cu pas (fiecare monom pe rand), atunci constructorul clasei va fi gol, neavand parametrii sau metode implementate in el, fiind utilizat doar pentru a genera un nou obiect, instanta a clasei Polinom.

public Polinom(){  
}

Cum am precizat anterior, un polinom fiind instantiat gol, trebuie sa ii fie adaugate monoame. Acest lucru este realizat prin metoda urmatoare:

public void addTermen(Monom x){  
 if(x.getCoeficient() != 0){  
 if(!calcPolinom(x)) this.termeni.add(x); //adaugam un nou termen polinomului final  
 Collections.*sort*(termeni, new CompPutereMonom());  
 }  
}

Metoda prezentata de adaugare in polinom primeste ca parametru un obiect, instanta a clasei Monom. Daca acesta are un coeficient nul, atunci adaugarea nu are sens. In caz contrar, se apeleaza metoda calcPolinom, iar daca rezultatul este fals, atunci se adauga noul termen. In final, indiferent de ce s-a ales, metoda sorteaza polinomul in ordinea descrescatoare a gradelor monoamelor sale.

Functia intermediara utilizata, calcPolinom, nu face decat sa ne usureze munca in adaugarea fiecarui monom.

public boolean calcPolinom(Monom x){  
 boolean state = false;  
 for (Monom m:this.termeni) {  
 if(m.getPutere() == x.getPutere()){  
 m.setCoeficient(x.getCoeficient() + m.getCoeficient());  
 state = true;  
 }  
 }  
 return state;  
}

Intr-un mod intuitiv matematicii clasice, aceasta metoda parcurge fiecare termen al polinomului si cauta un monom de grad egal cu cel trimis ca si parametru. In cazul in care se gaseste acest lucur, termenul deja existent in polinom va avea gradul actualizat, fapt ce evita marirea dimensiunii listei cand nu este necesar. Daca nu s-a realizat aceasta operatie cu succes (state = false), acest lucru se transmite functiei de adaugare termen, ce plaseaza noul monom in polinomul dorit, deoarece gradul sau nu mai este intalnit nicaieri.

Asadar, utilizand aceste doua metode auxiliare, adunarea si scaderea polinoamelor se poate realiza foarte usor, apeland doar metoda addTermen, in cazul adaugarii si addTermen pentru coeficient cu semn schimbat in cazul scaderii, functiile de adaugare parcurgand si modificand polinoamele.

//functia de adunare a polinoamelor

public void adunaPolinom(Polinom p){  
 for (Monom m: p.termeni) {  
 this.addTermen(m);  
 }  
}

//functia de scadere a polinoamelor  
public void scadePolinom(Polinom p){  
 for (Monom m: p.termeni) {  
 m.setCoeficient(-m.getCoeficient());  
 this.addTermen(m);  
 }  
}

Restul metodelor din clasa sunt implementate utilizand conventiile matematice pentru realizarea operatiilor respective, codul fiind pur si simplu o traducere din matematica in limbaj de programare, in cazul fiecarei operatii in parte.

Operatia de inmultire:

public void multPolinom(Polinom p){  
 List<Monom> rez = new ArrayList<Monom>();  
 for (Monom m1: this.termeni) {  
 for (Monom m2: p.termeni) {  
 rez.add(new Monom(m1.getCoeficient() \* m2.getCoeficient(),m1.getPutere() + m2.getPutere()));  
 }  
 }  
 this.termeni = rez;

Collections.*sort*(termeni, new CompPutereMonom());  
}

Operatia de imparire:

public ArrayList<Polinom> divPolinom(Polinom p){  
 ArrayList<Polinom> rezultat = new ArrayList<>();  
 Polinom cat = new Polinom();  
 Polinom rest = this;  
 Monom interm;  
 if(this.grad() < p.grad()) //se imparte polinomul mai mare la cel mai mic  
 p.divPolinom(this);  
 else{  
 interm = new Monom(this.termeni.get(0).getCoeficient() / p.termeni.get(0).getCoeficient(), this.termeni.get(0).getPutere() - p.termeni.get(0).getPutere());  
 cat.addTermen(interm);  
 p.multPolinom(cat);  
 rest.scadePolinom(p);  
 if(rest.grad() < p.grad()) {  
 rezultat.add(rest);  
 rezultat.add(cat);  
 return rezultat;  
 }  
 this.divPolinom(p);  
 }  
 return rezultat;  
}

Operatia de integrare:

public void deriveaza(){  
 for (Monom m: this.termeni) {  
 m.setCoeficient(m.getCoeficient() \* Math.*abs*(m.getPutere()));  
 m.setPutere(m.getPutere() - 1);  
 }  
}

Operatia de derivare:

public void integrare(){  
 for (Monom m: this.termeni) {  
 int putereImpartire = Math.*abs*(m.getPutere()+1); //pentru a evita division by 0  
 if(putereImpartire != 0){  
 m.setCoeficient((m.getCoeficient()) / Math.*abs*(m.getPutere()+1));  
 }  
 m.setCoeficient((m.getCoeficient()));  
 m.setPutere(m.getPutere() + 1);  
 }  
}

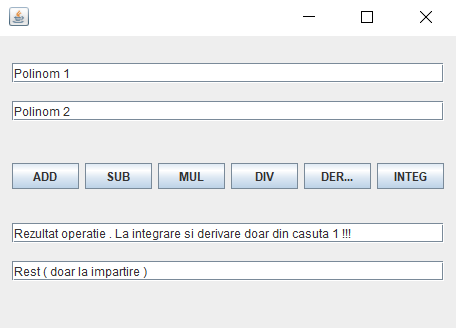
Toate aceste operatii efectueaza modificari ale coeficientilor din polinoame.

!! ATENTIE: Metodele de operatie pe polinoame sunt implementate ca metode void, fara a avea o valoare returnata; asadar, prin modul in care au fost implementate, ele necesita crearea unui obiect de tip polinom si apelarea lor utilizant obiectul de acel tip, de exemplu: P.integrare() realizeaza integrarea polinomului P, rezulatul operatiei fiind stocat in acesta.

## Clasa Window (interfata grafica):

Interactiunea dintre utilizator si implementarea proiectului este realizata printr-o interfata grafica. Aceasta se deschide in momentul in care programul este rulat.

Ea este cat se poate de intuitiva, fiind compusa din 4 campuri de text (2 pentru introducerea polinoamelor si 2 pentru afisarea rezultatului operatiei dorite) si 6 butoane (cu etichete specifice, ce denota calculul realizat), fiecare realizand o alta operatie.

Mai mult de atat, campurile de text ale interfetei contin, in starea initiala lor, mesaje ce au rolul de a-i sugera utilizatorului modul in care GUI-ul va trebui folosit, incat rezultatul reprodus sa fie cel asteptat si corect. 

In campurile de input se vor introduce, de la tastatura sub forma de string-uri, polinoamele. In momentul in care se detecteaza introducerea unui string, programul il analizeaza si parseaza pentru ca polinomul sa fie adus la o forma inteleasa de cod. Acest lucru se face folosind expresiile regulate (Regex), pe urmatorul pattern:

Pattern p = Pattern.*compile*("((-?\\d+(?=x))?(-?[xX])(\\^(-?\\d+))?)|((-?)[xX])|(-?\\d+)");

Cand un buton este apasat (adica se doreste efectuarea unei operatii pe unul / cele doua polinoame introduse in campurile specifice, sunt apelate functiile de get, ce initializeaza memoria cu 2 obiecte, instante ale clasei Polinom, echivalente cu input-ul primit de la utilizator. Deasemenea, aceste 2 functii au rolul important de a verifica si a valida string-ul introdus, moment in care un mesaj sub forma unei casute interactive va fi afisat, in cazul in care string-ul nu a fost corect.

public void getP(){  
 try{  
 p = getPolinom(jTextField1);  
 }  
 catch (Exception e){  
 JOptionPane.*showMessageDialog*(this, "Polinom 1 invalid introdus");  
 }  
}  
public void getQ(){  
 try{  
 q = getPolinom(jTextField2);  
 }  
 catch (Exception e){  
 JOptionPane.*showMessageDialog*(this, "Polinom 2 invalid introdus");  
 }  
}

## Clasa Main:

Aceasta este cea mai succinta clasa din tot proiectul, scopul ei fiind doar de a initializa fereastra interactiva pentru utilizator, restul operatiilor fiind facute si implementate in clasele aferente (inclusiv testarea).

public class MainClass {  
 public static void main(String[] args) {  
 Window gui = new Window();  
 gui.setVisible(true);  
 }  
}

# Testare:

Pentru partea de testare, in proiect a fost implementata o clasa separata, numita TestClass, ce utilizeaza pentru testele sale junit, mai exact:

import junit.framework.TestCase;

In cod sunt implementate atat functii de setup, cat si de teardown si test. Se testeaza in total 4 operatii: adunare, scadere, integrare, derivare, dorind verificarea corectitudinii acestora. Toate testele, mai putin unul trebuie sa rezulte ca fiind valide (fapt facut din adins pentru verificarea corectitudinii):

*assertEquals*(Q.toString(), rezultat.toString()); - test care da fail

# Concluzii si dezvoltare ulterioara:

Acest proiect pot spune ca m-a ajutat foarte mult in intelegerea programarii obiectuale, lucrand foarte mult cu instante ale diferitor clase definite. Unul dintre cele mai dificile lucruri, in opinia mea, l-a reprezentat partea de parsare a string-ului primit ca si input de la utilizator, intr-o forma valida pentru a fi recunoscuta de implementarea realizate a polinoamelor. Astfel, folosind pentru prima data expresiile regulate, am fost nevoit sa aloc timp suplimentar intelegerii acestei abordari, ce s-a finalizat din fericire cu succes.

Din punct de vedere al dezvoltarii ulterioare, proiectul poate fi imbunatatit implementand corect operatia de diviziune a polinoamelor, ea aruncand o exceptie si nefiind adaugata in proiect (in interfata grafica, aceasta functionalitate nu este prezenta => aspectul diferit al butonului). Mai mult de atat, sunt sigur ca anume cazuri in operatiile efectuate se poate sa nu fi fost tratate, nestiind de existenta lor, fapt ce constituie o imbunatarie posibila a programului realizat.

# Bibliografie:

In mare parte proiectul nu a fost conceput utilizand alte surse de informare directe, insa anumite metode de calcul din interiorul sau au avut la baza idei regasite in platformele online: youtube.com, pe canalul utilizatorului TheCodingTrain, si stackoverflow.com, unde au fost gasite exceptii sau probleme pe care si alti utilizatori le-au avut in dezvoltarea programelor lor.