

**FACULTATEA DE AUTOMATICA SI CALCULATOARE CATREDA CALCULATOARE**

**Calculator de polinoame**

Documentatie

Student:Ilies Oana-Elena

Grupa 30222 | An 2 semestrul 2

Cuprins

1.Obiectiv

2.Studiul problemei

3.Implementare

3.1 Diagrama UML

3.2 Clase

3.3 Metode

3.4 GUI

4.Testare

5.Concluzii si dezvoltari ulterioare

6.Bibliografie

1.Obiectiv

Acest sistem a fost conceput pentru a oferi o soluție eficientă și precisă pentru efectuarea diferitelor operații pe polinoame, utilizând o structură de date de tip map. Această implementare utilizează un map pentru a stoca cheia ca un intreg și valoarea ca un monom, permițând astfel manipularea și accesarea polinoamelor într-un mod eficient și ușor de urmărit. Interfața grafică a sistemului a fost concepută pentru a fi ușor de utilizat și intuitivă, permitând utilizatorilor să introducă polinoame cu coeficienți de tip double și să primească rezultate precise.

2.Studiul Problemei

Un polinom este o expresie matematică formată dintr-o sumă finită de termeni, fiecare termen fiind un produs al unei variabile cu un coeficient. De obicei, variabila este notată cu x, iar coeficienții sunt numere reale sau numere complexe.

Un monom este un termen simplu format din produsul unei variabile ridicate la o putere (exponent) și a unui coeficient, care poate fi o valoare numerică sau o variabilă.

Pentru separarea de string-uri s-a folosit o expresie *Regex* de forma:

"([+-]?\\d\*\\.?\\d\*)?x(\\^(\\d+))?"

*Care accepta un monom de tip:coeficient + „x” + „^” + exponet. Oricare alta forma nu e permisa, dar unele cazuri pot functiona corect. De exeplu pentru un polinom ce are exponentul 0, adica un numar real oarecare, este necesar introducerea unui polinom de forma: n(coeficient)x^0.*

Această expresie regulată este utilizată pentru a identifica și a extrage monoame dintr-un șir de caractere. Expresia regulată începe cu "([+-]?", care specifică că monomul poate începe cu un semn pozitiv sau negativ. "?" specifică că caracterul anterior (semnul) poate apărea sau nu. Următorul caracter, "\d\*", identifică zero sau mai multe cifre (deci coeficientul poate fi absent sau poate fi orice număr întreg). Caracterul "\^" indică că urmează un exponent, iar "(\d+)" identifică numărul real al exponentului.

Punctul, "\.", specifică că poate exista un punct în coeficient, pentru a indica zecimale.În continuare, apare "\d\*", care identifică orice cifră după punctul zecimal, în cazul în care există."?" apare din nou, pentru a indica că coeficientul poate fi absent.Înainte de variabila x apare "(\^(\d+))?", care specifică că puterea lui x poate fi prezentă sau nu.

**Opreratii:**

-> *Adunarea* și *scăderea* polinoamelor presupun adăugarea sau scăderea termen cu termen a monomilor cu aceeași putere. Dacă există termeni cu aceeași putere în ambele polinoame, atunci coeficienții lor se adună sau scad.

-> *Înmulțirea* polinoamelor implică înmulțirea fiecărui monom din primul polinom cu fiecare monom din al doilea polinom și apoi adunarea monomilor similari prin adăugarea coeficienților lor.

-> *Derivarea* unui polinom se referă la calcularea derivatei polinomului în funcție de variabila de bază, adică x. Derivata unui monom se obține înmulțind coeficientul cu puterea și apoi scăzând puterea cu 1or.

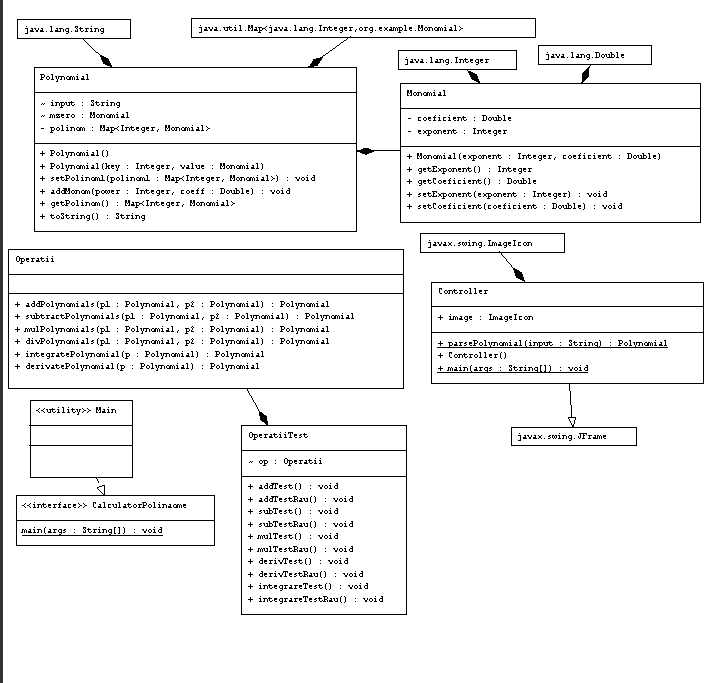
-> *Integrarea* unui polinom se referă la calcularea integralei definite sau indefinite a polinomului în funcție de variabila de bază, adică x. În general, există mai multe metode de integrare a unui polinom, inclusiv metode de integrare prin părți, integrare prin substituție și integrare prin fracții parțiale.

-> *Împărțirea*(neimplementata) polinoamelor implică împărțirea fiecărui termen din polinomul dividend cu primul termen din polinomul divisor și apoi înmulțirea restului obținut cu polinomul divisor și continuarea procesului până când nu mai este posibilă împărțirea. Restul obținut este restul împărțirii, iar partea întreagă obținută este formată din termenii rămași după procesul de împărțire.

3.Implementare

3.1.Diagrama UML

O diagramă UML (Unified Modeling Language) este o modalitate de a reprezenta grafic diferite aspecte ale unui sistem software sau al unei aplicații, folosind un set de simboluri și reguli standardizate. UML este o limbaj de modelare vizuală care poate fi utilizat pentru a defini, specifica, proiecta și documenta diferite tipuri de sisteme software.



3.2.Clase

a)Monomial

->Această clasă reprezintă un monom, care este o parte fundamentală a unui polinom. Aceasta are două atribute: exponentul (care este un număr întreg ce indică puterea variabilei în monom) și coeficientul (care este un număr real ce indică multiplicatorul variabilei în monom). De asemenea, clasa include metode de acces și modificare a acestor atribute.

b)Polynomial

->Această clasă implementează un polinom, reprezentat ca o colecție de monoame, folosind un map pentru a stoca coeficienții și exponentele acestora. Clasa conține metode pentru adăugarea de monoame la polinom, obținerea polinomului sub formă de map, precum și o metodă toString pentru afișarea polinomului într-un format ușor de citit. Polinomul este afișat în ordine descrescătoare a exponentului și monoamele cu coeficienți egali cu zero sunt ignorate.

c)Operatii

-> Această clasă definește o serie de metode pentru efectuarea de operații aritmetice asupra polinoamelor

d)Controller

-> Acest cod este o clasă Java care definește o interfață grafică de utilizator (GUI) pentru un calculator de polinoame. Clasa "Controller" este o clasă care extinde clasa "JFrame" și are o serie de elemente grafice precum butoane, etichete de text și câmpuri de text pentru a introduce polinoamele și pentru a afișa rezultatele

e)OperatiiTest

-> Aceasta este o clasă de testare unitară JUnit pentru a testa diferite metode ale clasei Operatii.

3.3.Metode

public static Polynomial parsePolynomial(String input)

\*Are rolul de a parsa un șir de caractere input pentru a construi și returna un obiect de tip Polynomial. \*Se creează un obiect nou de tip Polynomial pentru a adăuga monoamele găsite în șirul input. Se folosește un obiect *Pattern* pentru a căuta expresii regulate în șirul input. \*Se creează un obiect *Matcher* care va fi folosit pentru a găsi expresiile regulate în șirul input. Folosind metoda *find*() din obiectul Matcher, se găsesc toate aparițiile expresiei regulate în șirul input. \*Pentru fiecare monom găsit, se extrage semnul, coeficientul și exponentul. Semnul poate fi + sau -, coeficientul este un număr real, iar exponentul este un număr întreg pozitiv. Dacă coeficientul sau exponentul nu sunt specificate, se consideră că acestea sunt 1, respectiv 0. Dacă semnul este "-", coeficientul este înmulțit cu -1. \*Monoamele sunt apoi adăugate la obiectul Polynomial creat la început. La final, se returnează obiectul Polynomial creat cu monoamele găsite în șirul input.

public String toString()

\*Această metodă suprascrie metoda toString() a clasei Object și returnează o reprezentare sub formă de șir de caractere a polinomului. În această metodă, se iterează prin fiecare monom al polinomului și se construiește șirul de caractere corespunzător. Mai întâi, se sortează puterile monomiale în ordine descrescătoare și apoi se adaugă fiecare monom la șirul de caractere rezultat, în ordine descrescătoare a puterii. \*În cazul în care coeficientul monomului este pozitiv, se adaugă semnul de plus între monomuri, iar dacă este negativ, se adaugă semnul de minus. Dacă coeficientul este egal cu 1 sau -1, nu se afișează coeficientul în șir, iar dacă puterea este mai mare decât 1, se adaugă simbolul "x" și puterea corespunzătoare. În cazul în care coeficientul nu este 1 sau -1, se afișează coeficientul în șir. Pentru a limita numărul de zecimale al coeficientului, se utilizează formatul %.3f. Dacă polinomul este gol, se returnează șirul "0".

public Polynomial addPolynomials(Polynomial p1, Polynomial p2)

\*Această funcție primește ca parametrii două obiecte de tip Polynomial, p1 și p2, și returnează un nou obiect Polynomial care reprezintă suma celor două polinoame. \*În implementarea sa, se parcurg toți monoamele din primul polinom (p1) utilizând metoda *entrySet*() care returnează un Set de obiecte Map.Entry. Pentru fiecare monoam din p1, se extrage puterea, coeficientul și se verifică dacă polinomul p2 conține un monoam cu aceeași putere. În caz afirmativ, se adaugă coeficientul monoamelor, în caz contrar, se adaugă monoamul la rezultatul final. \*După aceea, se parcurge al doilea polinom (p2) și se adaugă toate monoamele care nu au fost luate în considerare în primul for. Metode *remove*()elimină monomul cu puterea power din p2 după ce s-a adăugat coeficientul monomului din p1 la coeficientul monomului din p2 la aceeași putere power. Dacă rezultatul final este un polinom vid, se returnează un polinom cu valoarea 0.

public Polynomial subtractPolynomials(Polynomial p1, Polynomial p2)

\*Se parcurg monomii primului polinom, iar pentru fiecare monom se verifică dacă există un monom cu același exponent în al doilea polinom. În caz afirmativ, coeficientul monomului din primul polinom este scăzut cu coeficientul monomului corespunzător din al doilea polinom, iar monomul din al doilea polinom este eliminat. Apoi se adaugă la rezultat monomul din primul polinom cu coeficientul actualizat. În cele din urmă, se adaugă la rezultat toți monomii rămași din al doilea polinom, cu coeficientul schimbat de semn (pentru a fi scăzuți). Dacă rezultatul este un polinom gol (fără monomi), se returnează un polinom gol creat.

public Polynomial mulPolynomials(Polynomial p1, Polynomial p2)

\*În interiorul primei bucle for se iterează peste toate monoamele din primul polinom (p1), iar în interiorul celei de-a doua bucle for se iterează peste toate monoamele din al doilea polinom (p2). Pentru fiecare pereche de monoame se calculează noul exponent (power) și noul coeficient (coeficient) al monoamelor rezultate din înmulțirea celor două monoame.

\*Apoi, se verifică dacă monoamul rezultat cu exponentul power există deja în polinomul rezultat. Dacă acesta există, se adaugă la coeficientul său coeficientul calculat anterior. În caz contrar, se adaugă monoamul rezultat în polinomul rezultat.

public Polynomial derivatePolynomial(Polynomial p)

\*Se parcurge fiecare termen al polinomului dat ca parametru utilizând un obiect de tip Map.Entry<Integer, Monomial>. Pentru fiecare termen se obține puterea și coeficientul și se verifică dacă puterea este mai mare decât 0 (adică, dacă există termeni în polinomul derivat). \*Dacă puterea este mai mare decât 0, se calculează noul coeficient prin înmulțirea vechiului coeficient cu puterea și se adaugă un nou termen cu puterea redusă cu 1 și noul coeficient în obiectul result.

public Polynomial integratePolynomial(Polynomial p)

\*metoda iterează prin fiecare monom din polinomul de intrare. Pentru fiecare monom, se calculează coeficientul noului monom prin împărțirea coeficientului monomului curent la puterea sa plus unu. Dacă puterea este mai mare sau egală cu zero, noul monom este adăugat la polinomul rezultat, cu puterea mărită cu unu și coeficientul calculat mai sus.

3.4.GUI



*Interfața* *grafică* *cu* *utilizatorul* (GUI) este o modalitate prin care utilizatorul poate interacționa cu un program de calculator prin intermediul unor componente vizuale, cum ar fi butoane, casete de text, meniuri și ferestre. GUI-urile sunt utilizate în mod obișnuit pentru a face aplicațiile mai accesibile și mai ușor de utilizat pentru utilizatorii finali, oferindu-le o modalitate intuitivă de a interacționa cu software-ul și de a realiza sarcini. Acestea sunt folosite într-o varietate de aplicații, de la aplicații de desktop și jocuri, la aplicații web și mobile. Interfața *JFrame* face parte din biblioteca grafică Swing din Java și oferă o fereastră care poate fi folosită ca container pentru alte componente grafice.

*Frame*=o fereastră grafică rectangulară pe care o aplicație o afișează pe ecranul unui computer. Această fereastră poate conține componente grafice, cum ar fi butoane, casete de selectare, meniuri și câmpuri de text, care permit utilizatorului să interacționeze cu aplicația.

*Butoane* ->button1(Adunare)

->button2(Inmultire)

->button3(Integrare)

->button4(Scadere)

->button5(Impartire)

->button6(Derivare)

*TextField* = spatii dreptunghiulare in care se pot introduce date de la tastatura. Dar pe langa asta pot fi folosite si pentru a afisa rezultatul fara a se putea introduce date de la tastatura.

textField1->aici se introduce primul polinom

textField2->aici se introduce al doilea polinom

textFieldRezultat->aici se afiseaza rezultatul

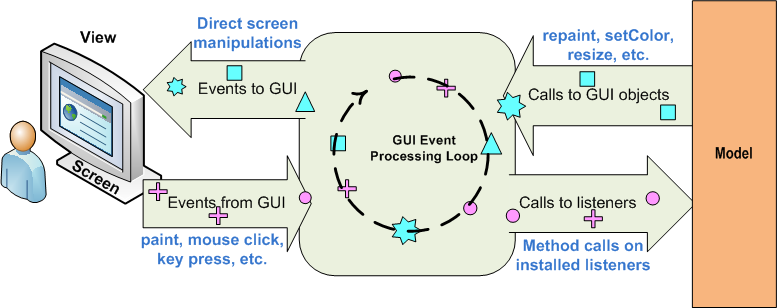
Clasa Controller are un constructor responsabil pentru aceasta implementare:

->Acest constructor este pentru crearea și inițializarea unei interfețe grafice de utilizator (GUI) pentru un calculator polinomial. În acesta, sunt setate dimensiunile ferestrei, sunt adăugate elemente grafice precum etichete, câmpuri de introducere a datelor și butoane pentru operații matematice. De asemenea, se definesc și evenimentele care au loc atunci când butoanele sunt apăsate, cum ar fi preluarea datelor introduse în câmpuri și apelarea funcțiilor corespunzătoare pentru efectuarea operațiilor matematice pe polinoamele introduse de utilizator.

\*Un *ActionListener* este o interfață definită în Java, care are o singură metodă abstractă numită *actionPerformed*(). Această metodă este apelată atunci când are loc un eveniment acționat, cum ar fi un clic pe un buton sau o selectare dintr-o listă.

\*Fiecare buton are un ActionListener atașat, care este definit în cadrul codului prin intermediul unei clase anonime. Atunci când un buton este apăsat, se va apela metoda actionPerformed() definită în clasa anonimă corespunzătoare butonului. În cadrul acestei metode se vor prelua datele introduse în câmpurile de text, se vor prelucra folosind operațiile specificate, și se va afișa rezultatul în câmpul de text corespunzător.

*Label* = este efectiv o etichetă, care poate fi titlul, informații, indicații etc. care pot ajuta utilizatorul să folosească progamul.



4.Testare

Pentru testare am ales un polinom mai complex pentu a arata capacitatea de functionare pe majoritatea cazurilor(cazuri de ineficienta inca n-au fost descoperite dar exista mereu posibilitatea)

Operatie adunare:



Operatie scadere:



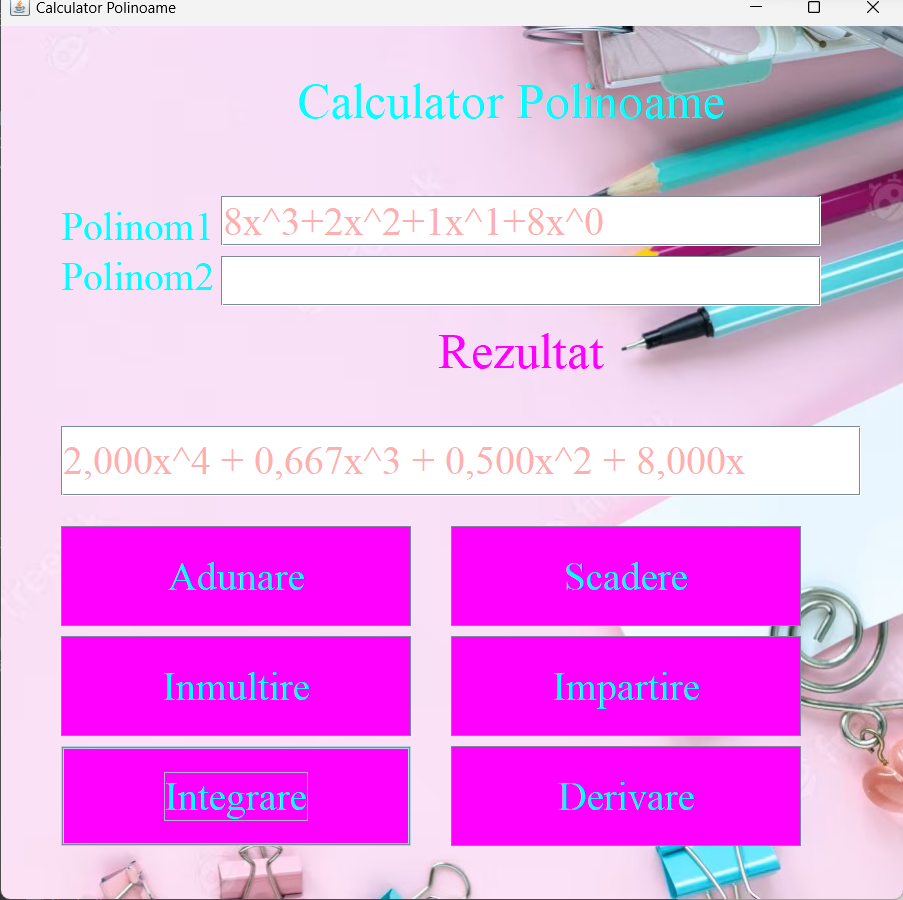
Observatie: in momentul in care polinomul are coeficientul 0 acesta nu mai este afisat. Este afisat doar in cazul polinomului cu exponent 0 ca „0”, de asemenea mai afiseaza „0” in unele cazuri de eroare, input incorect, metode neimplementata etc…

Operatie derivare:



Observatie:atat pentru operatia aceasta, cat si pentru cea de integrare se accepta introducerea polinomului doar in prima casuta: „Polinom1”.

Operatie integrare:



Observatie:Operatie nu afiseaza sub forma de fractie ci sub forma de numar real.

Petru operatia de inmultire am ales 2 polinoame mai scurte dar am pastrat complexitatea. Se vede ca daca monomul are acelasi grad coeficientii lor se aduna si formeaza un singur monom.



Cazul in care unul dintre coeficienti este 0.



**Teste din clasa OperatiiTest**

Am implementat pentru fiecare operatie cate 2 teste, unul passed si unul failed(intentionat)

Exemplu: test pentru operatii cu 1 polinom. Stare test failded. Cod asteptat: assertEquals(p1 .toString(), "4,000x + 2,000"); *Testrare derivare:*

@org.junit.Test  
public void derivTestRau() throws Exception  
{  
 Polynomial p1=new Polynomial();  
 p1.addMonom(2, 2.0);  
 p1.addMonom(1, 2.0);  
 p1.addMonom(0, -1.0);  
 p1=op.derivatePolynomial(p1);  
 System.*out*.println(p1.toString());  
 *assertEquals*(p1 .toString(), "4,000x + 2,000 + derivare");

}

Testare integrare: assertEquals(p1 .toString(), "0,667x^3 + x^2 - x"); passed assertEquals(p1 .toString(), "x^3/3 + x^2/1 - x.1"); failded

Exemplu: test pentru operatii cu 2 polinoame. Stare test passed *Test pentru adunare:*

@org.junit.Test  
public void addTest() throws Exception  
{  
 Polynomial p1=new Polynomial();  
 Polynomial p2=new Polynomial();  
 Polynomial r=new Polynomial();  
 p1.addMonom(2, 2.0);  
 p2.addMonom(2, 1.0);  
 p1.addMonom(1, 2.0);  
 p2.addMonom(1, 3.0);  
 p1.addMonom(0, -1.0);  
 p2.addMonom(0, 1.2);  
 r=op.addPolynomials(p1, p2);  
 System.*out*.println(r.toString());  
 *assertEquals*(r.toString(), "3,000x^2 + 5,000x + 0,200");  
  
}

*Test pentru scadere:* assertEquals(r.toString(), "x^2 - x - 2,200"); passed assertEquals(r.toString(), "x^2 + 2x - 2,200"); failed *Test pentru inmultire:* assertEquals(r.toString(), "2,000x^4 + 8,000x^3 + 7,400x^2 - 0,600x - 1,200");passed assertEquals(r.toString(), "2,123x^4 + 8,000x^3x - 1,200"); failed

5.Concluzii si dezvlotari ulterioare

În concluzie, calculatorul de polinoame prezentat în codul furnizat este un instrument util pentru efectuarea unor operații matematice cu polinoame. Acesta oferă o interfață grafică prietenoasă utilizatorului, care permite utilizatorului să introducă polinoame și să efectueze operațiile de adunare, scădere, înmulțire, derivare si integrare

O dezvoltare ulterioară posibilă ar fi de a extinde funcționalitatea programului prin adăugarea de funcții suplimentare cum ar fi găsirea rădăcinilor polinomului într-un anumit punct. De asemenea, se poate adăuga o opțiune de exportare a rezultatelor într-un fișier sau de a salva polinoamele într-o bază de date pentru a le putea recupera ulterior.

De asemenea, programul poate fi optimizat pentru a asigura o performanță mai bună în cazul în care se lucrează cu polinoame mari. De exemplu, putem adăuga o funcționalitate de cache pentru a stoca polinoamele și operațiile intermediare, astfel încât să nu fie nevoie să le recalculeze de fiecare dată când utilizatorul dorește să efectueze o operație.

În final, acest calculator de polinoame poate fi util pentru studenți, profesori sau persoane care lucrează cu polinoame în cadrul activităților lor de zi cu zi și poate fi extins și optimizat pentru a oferi o experiență mai bună utilizatorilor.

6.Bibliografie

<https://stackoverflow.com/>

<https://www.wikipedia.org/>

<https://www.w3schools.com/java/java_oop.asp>

<https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions>

<https://regexr.com/>