**DOCUMENTATIE TEMA 1**

**CALCULATOR POLINOAME**

**Ovreiu Auraș Dănuț**

**Grupa 30229**

**Profesor Laborator Assist Antal Marcel**

Contents

[1. Cerinte funcționale 3](#_Toc66905559)

[2. Obiective 3](#_Toc66905560)

[2.1. Obiectivul principal 3](#_Toc66905561)

[2.2. Obiective secundare 3](#_Toc66905562)

[3. Analiza problemei 3](#_Toc66905563)

[4. Proiectare 4](#_Toc66905564)

[4.1. Structuri de date 4](#_Toc66905565)

[4.2. Algoritmi 4](#_Toc66905566)

[4.3. Diagrame de clase 5](#_Toc66905567)

[5. Implementare 6](#_Toc66905568)

[5.1. Clase 6](#_Toc66905569)

[5.2. Metode 6](#_Toc66905570)

[6. Testare 10](#_Toc66905571)

[7. Concluzii și Dezvoltări Ulterioare 11](#_Toc66905572)

[8. Bibliografie 11](#_Toc66905573)

# 

# 1. Cerinte funcționale

La această temă se cere implementarea unui calculator de polinoame care să permită utilizatorului să selecteze operația dorită prin intermediul unei interfețe grafice și să vizualizeze rezultatul obținut. Polinoamele se obțin prin extragerea coeficienților și a puterilor dintr-un string care reprezintă polinomul, iar operațiile dorite sunt adunarea, scăderea, înmulțirea, împărțirea, derivarea și integrarea.

# 2. Obiective

## 2.1. Obiectivul principal

Obiectivul principal al acestei teme este proiectarea și implementarea unui sistem de procesare a polinoamelor de o singură variabilă cu coeficienți reali.

## 2.2. Obiective secundare

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Descrierea cerinței | se prezintă sumar funcționalitatea proiectului | 3 |
| Alegerea structurilor de date | se aleg structuri de date care permit crearea de monoame și polinoame într-un mod cât mai accesibil | 4 |
| Împărțirea pe clase | proiectul se împarte pe clase grupate in pachete respectând arhitectura model-view-controller | 4 |
| Dezvoltarea algoritmilor | se vor implementa algoritmi pentru realizarea operațiilor pe polinoame acolo unde este cazul | 4 |
| Implementarea soluției | se va descrie fiecare clasă cu metode si câmpuri importante, precum și implementarea interfeței utilizator | 5 |
| Testare | se verifică corectitudinea efectuării operațiilor prin realizarea unui test pentru fiecare operație | 6 |

# 3. Analiza problemei

În cadrul acestei teme sunt impuse o serie de specificații, după cum urmează:

* folosirea paradigmelor de programare orientată obiect(încapsulare, folosirea de clase aferente, denumirea claselor, metodelor, pachetelor, variabilelor în mod corect)
* utilizarea de liste în locul array-urilor
* folosirea buclei for-each în locul celei obișnuite
* realizarea unei interfețe grafice dedicate în care să fie posibilă vizualizarea efectuării operațiilor
* folosirea arhitecturii MVC
* implementarea operațiilor de adunare, scădere, împărțire, înmulțire, derivare, integrare
* folosirea de expresii regulate pentru extragerea coeficienților și a puterilor
* testarea unitară a operațiilor folosind Junit

Un polinom reprezintă o expresie, formată din mai multe monoame, care, la rândul lor sunt reprezentate prin coeficienți și exponenți. Există mai multe metode prin care poate fi realizat un sistem de calcul al polinoamelor.

Metoda folosită în acest caz este aceea de extragere a coeficienților și a puterilor unui polinom dintr-un string pe baza unui regex, care să permită separarea valorilor numerice necesare de restul conținutului string-ului. Stocarea coeficienților și a puterilor se realizează în câte o listă, iar citirea string-ului se va face direct din interfață, prin utilizarea de getteri și setteri în cadrul unor text-fielduri.

Această metodă este una relativ ușoară, accesibilă și elegantă, evitând o serie de operații pe string-uri a căror implementare ar fi fost de altfel destul de anevoioasă.

# 4. Proiectare

## 4.1. Structuri de date

Principala structură de date folosită este lista(List <Monom> polinom=new ArrayList<>()), aceasta fiind mai eficientă la operațiile de actualizare și tipărire a elementelor aflate pe anumite poziții cunoscute decât celelalte colecții. O altă structură creată pentru a reține mai ușor monoamele are următoarea formă:

public Monom(double coeficient, int putere){

this.coeficient=coeficient;

this.putere=putere;

}

Memorând coeficienții și puterile pentru fiecare monom putem compune cu ușurință un polinom prin simpla parcurgere a listei de monoame cu o buclă for-each.

## 4.2. Algoritmi

Principalii algoritmi sunt cei folosiți la împărțire și înmulțire.

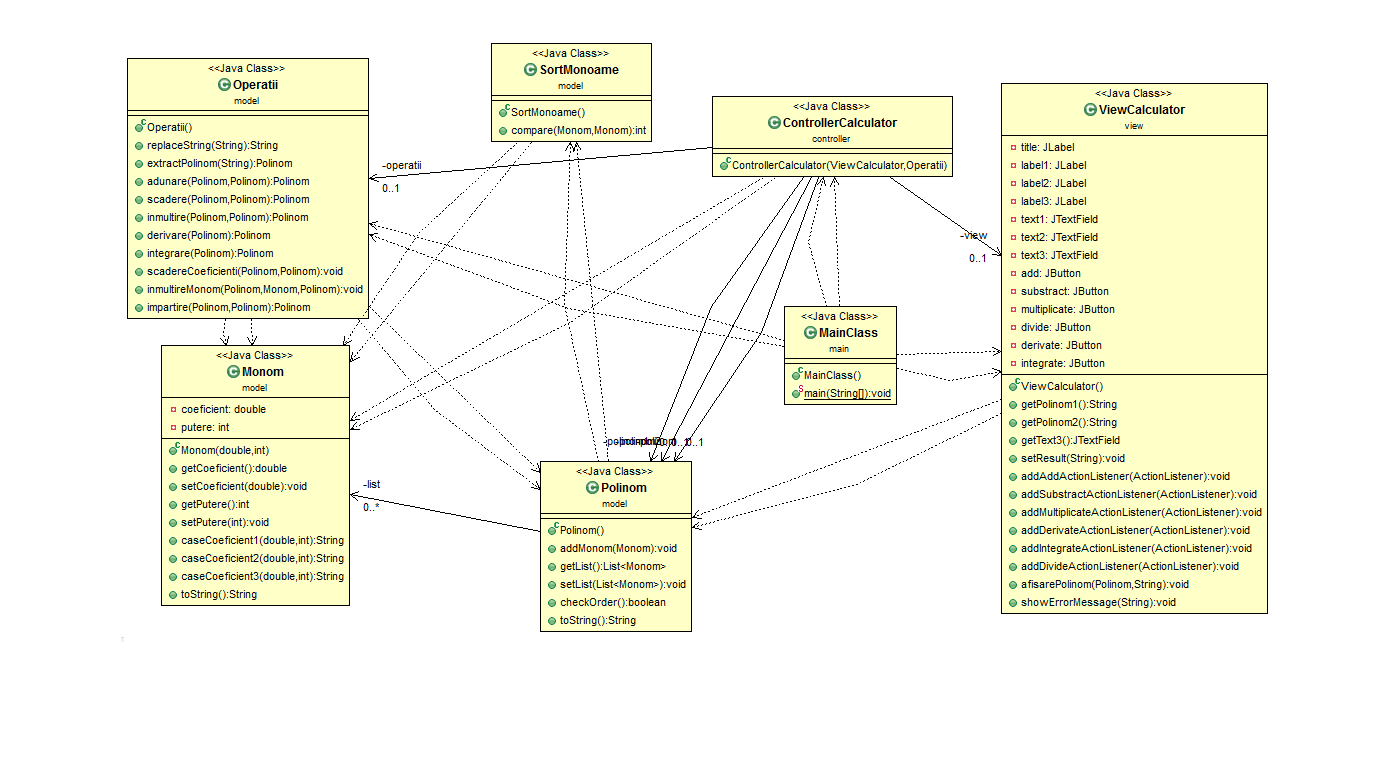
La polinoame algoritmul de împărțire este oarecum asemănător celui utilizat la împărțirea a două numere, în sensul că ne folosim de deîmpărțitor, împărțitor și cât. Inițial salvăm polinomul inițial în alt obiect, după care împărțim monomul cu gradul cel mai mare de la acel obiect cu monomul cu gradul cel mai mare de la împărțitor, înmulțim monomul obținut astfel cu împărțitorul și scădem din deîmpărțitor noul rezultat obținut. Pentru noul deîmpărțitor vom repeta pașii menționați până când gradul deîmpărțitorului va deveni strict mai mic decât cel al împărțitorului. Polinomul rezultat în urma operației de împărțire va fi lista de monoame obținute prin împărțiri successive.

La algorimtul de înmulțire luăm coeficienții de la cele două polinoame începând cu cei corespunzători celor mai mari puteri și îi înmulțim pe rând(cel de la puterea cea mai mare mai întâi cu fiecare de la al doilea polinom), iar puterile le adunăm. Când se ajunge la două sume de puteri identice păstrăm constantă noua putere formată și adunăm coeficienții aferenți celor două sume de puteri.

## 4.3. Diagrame de clase

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificații pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora ( numite și obiecte ). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT

.



# 5. Implementare

## 5.1. Clase

***Clasa ControllerCalculator:***

Această clasă este granița dintre interfata grafică și programul propriu-zis, unde se petrece interacțiunea utilizator-sistem de calcul.

***Clasa Operatii:***

Este clasa în care sunt implementate operații de adunare, scădere, înmulțire, împărțire, derivare, integrare, precum

și metode în care sunt prelucrate string-urile reprezentând polinoame pentru a extrage coeficienții și puterile.

***Clasa Main:***

Este clasa în care spunem programului ce să execute. În această clasă se realizează deschiderea interfeței grafice.

***Clasa Monom:***

Este clasa în care ne creăm monoamele.

***Clasa Polinom:***

Este clasa în care salvăm monoamele și creăm polinoame din acestea.

***Clasa SortMonoame:***

Este clasa în care sortăm monoamele după puterile acestora.

***Clasa ViewCalculator:***

Este clasa în care se creează interfața grafică în sine.

***Clasa TestOperații:***

Este clasa în care se testează corectitudinea efectuării operațiilor.

## 5.2. Metode

***Metode utilizate în clasa Monom:***

În această clasă se regăsesc metode pentru reținerea și modificarea coeficienților, respectiv a puterilor polionoamelor(getteri și setteri).

public double getCoeficient() {  
 return coeficient;  
}  
  
public void setCoeficient(double coeficient) {  
 this.coeficient = coeficient;  
}

public int getPutere() {  
 return putere;  
}  
  
public void setPutere(int putere) {  
 this.putere = putere;  
}

Tot în această clasă avem și metode asociate metodei toString, în care se realizează afișarea cât mai corectă a monomului în funcție de exponent(dacă este egal cu 0, egal cu 1 sau mai mare decât 1). În metoda toString aceste metode sunt apelate în mod explicit, la final returnându-se forma literară a monomului în funcție de exponentul său.

***Metode utilizate în clasa SortMonoame:***

Unica metodă a acestei clase este compare, care sortează monoamele dintr-o listă în ordinea descrescătoare a puterilor lor.

***Metode utilizate în clasa Polinom:***

În această clasă utilizăm o listă pentru memorarea monoamelor și ulterior crearea unui polinom.

public List<Monom> getList() {  
 return list;  
}  
  
public void setList(List<Monom> list) {  
 this.list = list;  
}

Pentru adăugarea unui monom în listă folosim următoarea metodă:

public void addMonom(Monom m){  
 this.list.add(m);  
}

Tot aici avem și o metodă prin care verificăm dacă monoamele sunt sortate în ordinea descrescătoare a exponentului lor. În cadrul acestei metode se returnează o valoare booleană(true dacă lista e sortată descrescător, altfel false).

La final apare din nou metoda toString, în care formăm expresia literară a polinomului prin concatenarea la sfârșitul unui string gol inițial a formei literare a monoamelor din listă.

***Metode utilizate în clasa Operatii:***

Pentru început, această clasă conține metode necesare pentru extragerea coeficienților și a puterilor dintr-un string.

Metoda replaceString are rolul de a aduce fiecare monom dintr-un polinom la forma ax^n, returnându-se string-ul modificat.

Metoda checkEqualDegrees verifică dacă nu cumva utilizatorul nu a introdus două sau mai multe monoame având același grad. În acest caz se calculează suma acelor monoame, în final rămânând numai unul singur cu acel exponent.

În continuare avem metoda extractPolinom, care după cum îi zice și denumirea are rolul de a extrage coeficienții și puterile dintr-un string. Aceasta se realizează folosind regex(regular expressions) pentru a separa într-un array de string-uri tot ce se află înainte sau după caracterele “+”, “-” și “x”, înainte de aceasta înlocuindu-se toate caracterele “^” cu “” folosind metoda replace pentru a facilita separarea coeficienților și a puterilor.

String numbers[]=polinom1.replace("^", "").split("((?=\\+)|(?=\\-)|x)");

După această prelucrare, ar trebui să obținem un array de string-uri reprezentând coeficienții și puterile. În unele cazuri particulare array-ul ar putea conține string-uri de forma “+1”, “+2.2”, fapt care ar constitui o problemă la parsare. Din acest motiv în caz că un element conține caracterul “+” îl vom înlocui cu “”.

În array-ul numbers[] coeficienții se află pe poziții pare, iar puterile pe poziții impare, astfel că putem introduce coeficienții într-o listă și puterile în altă listă în aceeași buclă.

List<Double> coef=new ArrayList<>();  
List<Integer> exp=new ArrayList<>();  
for(int i=0; i<numbers.length; i+=2){  
 numbers[i].replace("+", "");  
 coef.add(Double.*parseDouble*(numbers[i]));  
 exp.add(Integer.*parseInt*(numbers[i+1]));  
}

Pentru ca implementarea operațiilor să fie mai ușoară, adăugăm puteri și coeficienți nuli acolo unde nu există puterile respective. Cu alte cuvinte, vom avea monoame cu toate puterile de la 0 la gradul polinomului.

În această metodă am apelat metodele replaceString și checkEqualDegrees pentru a trata anumite cazuri particulare de scriere a polinomului în interfața grafică.

Metodele pentru operațiile de adunare, scădere, înmulțire și împărțire au fiecare câte două argumente(Polinom polinom1, Polinom polinom2), iar cele de derivare și integrare numai un argument(Polinom polinom1), aplicându-se doar primului polinom.

Metoda de adunare inserează monoamele din polinomul cu gradul mai mare în alt polinom inițial nul(listă de monoame), iar apoi parcurge cu o buclă for-each polinomul cu gradul mai mic, setând coeficienții polinomlui instanțiat inițial ca fiiind sumă dintre coeficienții primului și celui de-al doilea polinom, însă doar pentru coeficienții

care corespund puterilor identice cu cele din polinomul cu gradul mai mic. La final returnăm polinomul pe care l-am instanțiat inițial.

Metoda de scădere este asemănătoare celei de adunare, singura diferență fiind introducerea unei variabile denumite sign care poate fi 1 sau -1, în funcție de gradele celor două polinoame. În caz că primul polinom are gradul mai mare sign ia valoarea 1 și spre deosebire de adunare se scad coeficienții corespunzători puterilor identice, iar dacă primul polinom este cel cu gradul mai mic sign ia valoarea -1, coeficienții celui de-al doilea polinom schimbându-și semnul. În acest caz în bucla for-each vom aduna coeficienții puterilor identice(asta deoarece “-” cu “-” dă prin convenție de semn “+”.

La metoda de înmulțire instanțiem un nou polinom, iar în lista sa de monoame vom adăuga numai coeficienți egali cu 0 și puteri începând de la suma gradelor celor două polinoame până la 0. După aceea, vom parcurge cele două liste de polinoame prin două bucle for-each imbricate, în care vom adăuga la polinomul instanțiat inițial produsul coeficienților monoamelor celor două polinoame primite ca parametri, începând cu cele mai mari grade, suma puterilor acestor monoame fiind de fiecare dată identică cu puterea monomului polinomului intanțiat inițial la care am ajuns în iterație.

La metoda de derivare polinomul rezultat se obține simplu, printr-o parcurgere for-each a monoamelor polinomului primit ca parametru. Noii coeficienți vor fi egali cu produsul dintre coeficienții inițiali și exponenții fiecărui polinom, iar puterea va scădea cu 1 la fiecare monom, mai puțin la monomul cu puterea 0 care va rămâne la fel.

La metoda de integrare se vor modifica doar acele monoame care au coeficienții diferiți de 0, la celelalte coeficienții devenind raportul dintre coeficienții inițiali și puterea adunată cu 1 pentru fiecare monom, iar puterile cresc fiecare cu 1.

La metoda de împărțire am aplicat algoritmul de împărțire menționat la capitolul 4.2. La această metodă am aruncat o excepție pentru cazul particular în care se încearcă împărțirea cu polinomul nul. În această situație se va afișa atât la consolă, cât și pe interfața grafică un mesaj corespunzător.

if(polinom2.getList().get(0).getCoeficient()==0 && polinom2.getList().get(0).getPutere()==0){  
 throw new Exception("nu putem imparti cu polinomul nul");  
}

Tot la această metodă am apelat niște versiuni modificate ale metodelor de scădere și înmulțire, deoarece am avut probleme când am apelat direct metodele originale în metoda de împărțire.

***Metode utilizate în clasa ViewCalculator:***

În această clasă metodele utilizate sunt scurte, folosite pentru adăugare de listeneri pentru butoane, getteri și setteri pentru text fielduri, pentru afișarea polinomului rezultat și respectiv pentru afișsarea mesajelor de eroare pe interfață.

Această clasă răspunde de afișarea interfeței grafice, având o serie de elemente caracteristice acesteia.

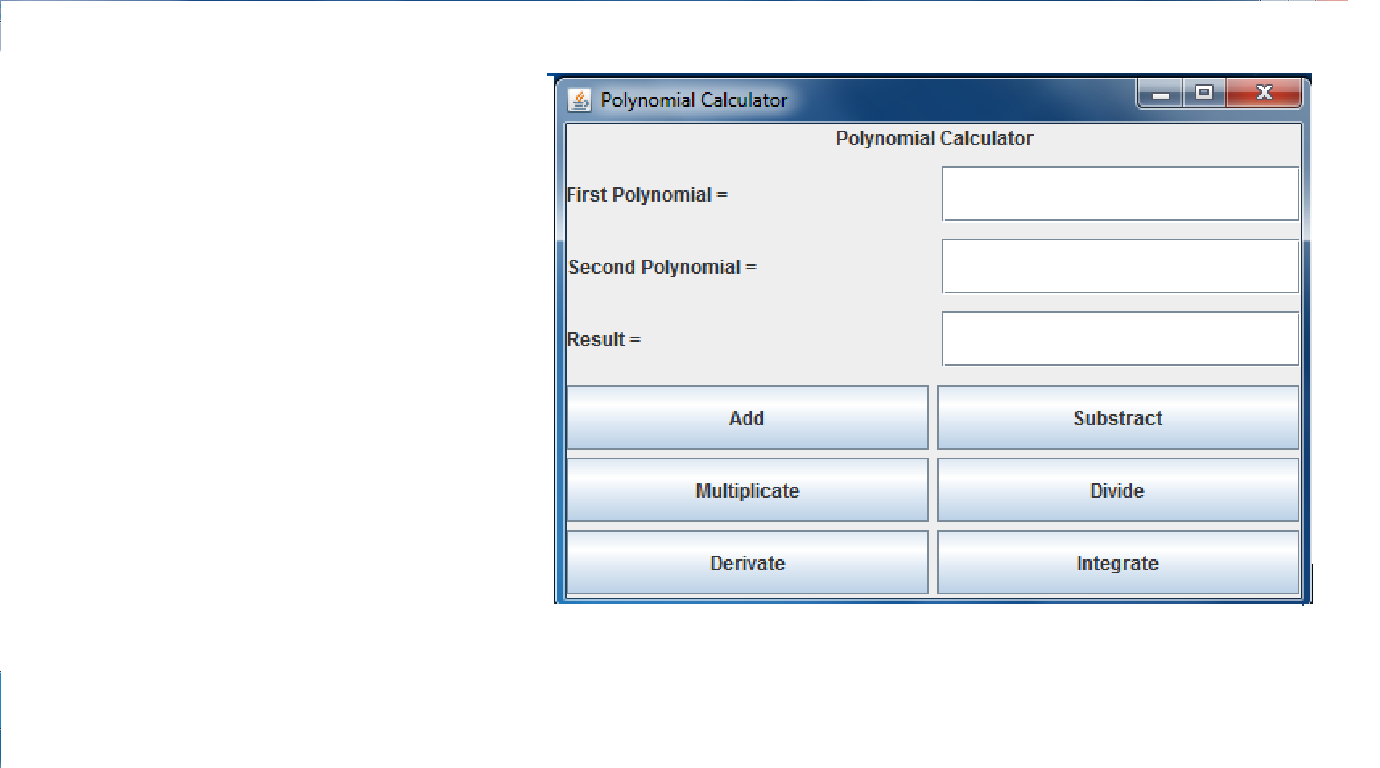
Principalele câmpuri ale clasei sunt label-uri, textfield-uri, și butoane.

Label-ul title este utilizat pentru a afișa titlul proiectului pe interfața grafică, iar label1, label2 și label3 pentru a arăta ce polinoame vor fi afișate în dreptul textfield-urilor.

Textfield-urile reprezintă un spațiu inițial gol în care putem scrie polinoamele inițiale(primele 2 textfield-uri), iar cel de-al treilea este cel în care se va afișa polinomul rezultat în funcție de operația selectată.

Cele 6 butoane prezente pe interfața grafică sunt folosite pentru a selecta operația dorită.

În principiu, label-urile, textfield-urile și butoanele au fost grupate pe interfața grafică utilizând GridLayout, cu excepția label-ului pentru titlu la care s-a folosit BoxLayout. La final panel-urile rezultate au fost grupate folosind BoxLayout.



***Metode utilizate în clasa ControllerCalculator:***

În această clasă se implementează metodele pentru operații menționate în clasa ViewCalculator. Fiecare astfel de metodă are un ActionListener care este activat la fiecare click pe un buton.

În fiecare dintre aceste metode sunt prinse excepții și sunt afișate mesaje de eroare în interfața grafică și chiar și în consolă(atunci când împărțim cu polinomul nul). Mesajele afișsate sunt referitoare la erori de sintaxă(de exemplu, utilizatorul introduce mai multe variabile, nu pune cifrele unde trebuie, folosește alte caractere decât cele premise) și la modul în care sunt introduse monoamele(se introduce un polinom având monoamele neordonate descrescător după putere).

# 6. Testare

Pentru verificarea corectitudinii efectuării operațiilor am testat proiectul utilizând Junit.

Pentru fiecare operație am făcut câte un test, iar la final toate testele s-au efectuat cu succes.

De exemplu, pentru adunare am ales mai întâi polinoamele x^3+x+2 și 3x-2. Coeficienții rezultați în urma acestei operații, începând de la puterea cea mai mare ar trebui să fie 1, 0, 4 și 0, iar la eroarea relativă(delta) am pus 0.01, coeficienții fiind de tip double. Alt scenariu de testare a fost pentru polinoamele –x+2 și x-2, rezultatul așteptat fiind 0 și 0, ceea ce s-a și întâmplat.

Alt exemplu de testare ar fi la operația de înmulțire, la care am luat inițial polinoamele 2x^2+3 și 3x-2. Coeficienții la care ne-am aștepta sunt 6, -4, 9 și -6, iar puterile 3, 2, 1 și 0.

Un alt exemplu este împărțirea, unde am luat mai întâi polinoamele x^3-7 și x-1, coeficienții așteptați fiind 1, 1, 1, iar puterile 2, 1, 0. Am testat aici și cazul în care se încearcă împărțirea cu polinomul nul, pentru polinoamele x-1 și 0, în acest caz aruncându-se excepția dorită.

# 7. Concluzii și Dezvoltări Ulterioare

În concluzie, sunt de părere că acest proiect mi-a reamintit paradigmele poo și cunoștințe din programarea în java, precum și modul de creare a unei interfețe grafice în GUI.

Ca dezvoltări ulterioare, consider că acest calculator de polinoame ar putea fi extins cu mai multe operații(de exemplu, ridicarea la putere), iar operațiile să fie efectuate pe numere complexe, nevoia de calcul rapid și eficient fiind din ce în ce mai mare.

# 8. Bibliografie

1. [*https://users.utcluj.ro/~igiosan/teaching\_poo.html*](https://users.utcluj.ro/~igiosan/teaching_poo.html)

2. Wikipedia