Sistem de procesare a polinoamelor de o singura variabila

Bilc Sergiu- Ion

Grupa 30224

**Cuprins**

1. Obiectivul temei
2. Analiza problemei
3. Proiectare
4. Implementare
5. Testare
6. Rezultate
7. Concluzii
8. Bibliografie
9. **Obiectivul temei**

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

***Obiective***: Se doreste crearea unei aplicatii Java, care sa realizeze o serie de operatii uzuale asupra unor polinoame cu coeficienti reali introduce de la tastatura de un posibil utilizator:

* Citirea si afisarea unui polinom
* Operatiile de adunare, scadere, inmultire si impartire a doua polinoame
* Operatii efectuate asupra unui singur polinom: derivarea si integrarea
* Realizarea unei interfete grafice implementata cu elemente Swing

1. **Analiza problemei**

Procesarea polinoamelor poate reprezenta o problema relativ complexa, deoarece, asupra acestora poate fi aplicata o gama extrem de larga de operatii, precum calculul valorii polinomului intr-un punct sau gasirea solutiilor polinomului. In proiectul de fata am decis sa implementez doar cateva dintre operatiile uzuale, mai exact: adunarea, scaderea, inmultirea, impartirea, derivarea si integrarea.

Rezolvarea propusa de mine are o complexitate medie si eu o consider a fi una clara, usor de asimilat. Am avut in vedere implementarea unei interfete cat mai placuta din punct de vedere vizual, bazata pe un mod de introducere a polinoamelor cat mai putin anevoios utilizatorului, dar eliminand in acelasi timp posibilitatea ca acesta sa introduca date eronate.

Pentru reprezentarea polinoamelor am decis sa merg pe varianta in care fiecare polinom este reprezentat de un ArrayList de monoame, un monom fiind caracterizat de 2 termeni reali, mai exact, coeficientul si puterea variabilei. Am ales aceasta reprezentare deoarece o consider a fi una mai usor de modelat in interfata grafica, in timp ce datele sunt structurate eficient si logic in memorie, fiind usor de operat cu acestea in vederea implementarii operatiilor. Am decis sa folosesc doar date de tipul real deoarece in cazul anumitor operatii, rezultatele calculelor asupra unor numere intregi ar avea ca rezultat numere reale, lucru ce ar putea cauza inconveniente pe parcursul proiectului. Am eliminat astfel inca de la inceput aceasta problema si am reusit in acelasi timp sa pastrez si un aspect placut al aplicatiei, lucru care o sa il prezint mai jos.

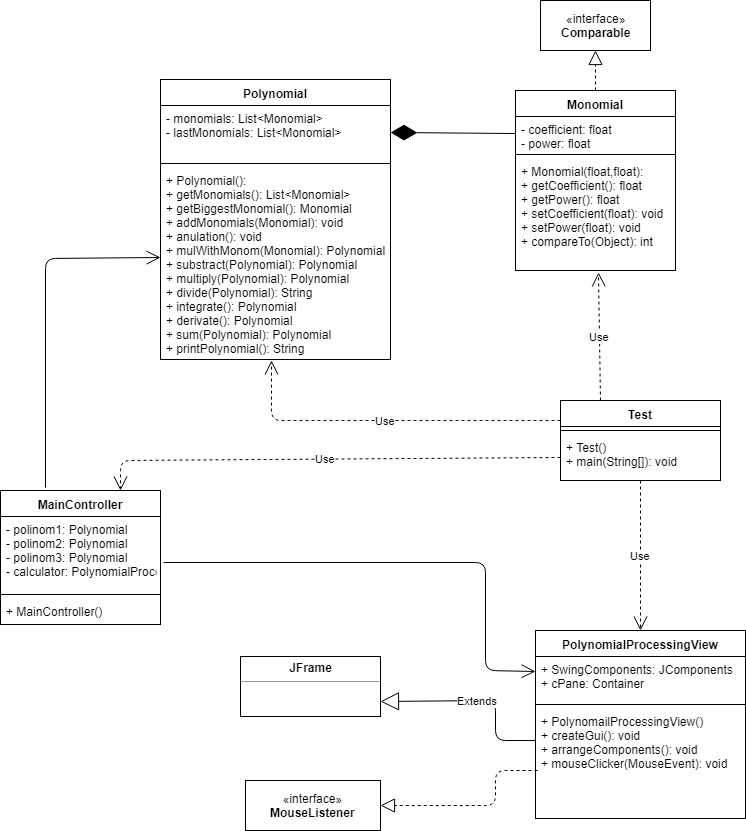
Din punct de vedere matematic, un polinom reprezinta o suma de monoame, acestea fiind formate, la randul lor, dintr-o constanta reala numtia coeficient si o variabila care detine un exponent constant pozitiv, denumit putere. Exponentul variabilei din monom este egal cu gradul variabilei in acel polinom. Gradul unui termen constant este zero, iar gradul unei variabile fara exponent este unu. Astfel, polinomul se poate scrie sub forma:

P(X) = an Xn + an-1Xn-1 + ... + a2X2 + a1X + a0, unde a0,a1, ..., an  sunt coeficientii polinomului, iar 1,2, ..., n reprezinta exponentii variabilei.

Pentru rezolvarea operatiilor m-am folosit de faptul ca suma, adunarea, inmultirea, impartirea, derivarea si integrarea unor/unui polinom vor avea care rezultat tot un polinom.

In cele ce urmeaza o sa prezint cum am decis ca utilizatorul sa introduca datele de intrare impreuna cu interfata grafica, cum salvez polinoamele astfel incat sa le pot prelucra in continuare si cum am efectuat operatiile asupra acestor polinoame.

1. **Proiectare**
   1. **Diagrama UML**



* 1. **Modelare**

Am structurat proiectul folosindu-ma de arhitectura MVC (Model-View-Controller). Astfel, aplicatia este aranjata in 3 pachete: Models(care reprezinta datele, obiectele, cu care se lucreaza: Polynomial si Monomial), Views („vederile”, sectiunea de interfata grafica), si Controllers ( controlorul care preia intrarile de la utilizator si le transpune in modelele cu care lucram, mai apoi ocupandu-se si de operatiile efectuate). Au rezultat astfel 5 clase:

* ***Clasa Polynomial***: utilizata pentru stocarea polinoamelor(Listelor de monoame), dar si pentru efectuarea operatiilor: de adunare, scadere, inmultire, derivare si integrare
* ***Clasa PolynomialProcessingView***: clasa in care am realizat interfata gafica
* ***Clasa Monomial***: clasa in care am creat propiu-zis obiectul monom, avand ca field-uri un coeficient si o putere, ambele de tip float.
* ***Clasa MainController***: clasa in care am realizat „controlul” aplicatiei, actiunile butoanelor, impreuna cu efectuarea operatiilor
  1. **Structuri de date**

Principala structura de date folosita este ArrayList-ul. Un ArrayList este o coloectie de elemente indexate intr-o anumita ordine, permitand in acelasi timp accesul rapid la date, dar o stergere si o insertie mai lenta. Aceasta clasa este continuta in spatiul System.Collections si ne permita sa construim un vector(o lista) de dimensiune variabila, la care putem adauga oricand noi elemente.

* 1. **Interfete folosite**

Pe parcursul proiectului am folosit trei interfete deja existente: Comparable,

ActionListener, MouseListener. Cea dintai este folosita pentru sortarea listelor de monoame, sortare efectuata in functie de exponentii monoamelor, si necesara pentru ca monoamele sa fie stocate si afisate pe ecran intr-o ordine logica.

1. **Implementare**

* **Clasa Monomial:**

Clasa Monomyal reprezinta structura de date de baza folosita in realizarea acestui proiect. Aceasta este o clasa simpla, care are doar doua atribute de tip flotant, mai exact, coefficient si power. Acesta clasa reprezinta transpunerea in programarea orientata obiect a celei mai simple unitiati din care este formata un polinom, si anume monomul. Pe parcursul proiectului am lucrat in permanenta cu obiecte de tip Monomial si mai ales cu Lista de obiecte Monomial.

* **Clasa Polynomial:**

Clasa principala a proiectului, in care se stocheaza listele de monoame, fiecare polinom fiind caracterizat de catre o astfel de structura(*monomials*). Tot in aceasta clasa am creat o alta lista de monoame (*lastMonomials*)pe care o folosesc la eliminarea ultimelor monoame adaugate in lista principala. Un aspect foarte important e faptul ca **introducerea monoamelor se face secvential.** (Un monom este introdus in lista la apasarea unui buton, iar apoi, daca se doreste, alt monom poate fi adaugat la lista prin apasarea aceluiasi buton).

**Metode importante:**

* ***getMonomilas(): List<Monomial>*** : returneaza lista de monoame prin care este caracterizat polinomul curent;
* ***addMonomials(Monomial): void*** : metoda de adaugare a unui monom la lista de monoame, cu anumite constrageri(ex: coeficientul monomului diferit de 0). Aceasta metoda insereaza monomul si in lista *lastMonomials,* iar dupa ce s-au realizat cele doua adaugari, metoda sorteaza lista principala(*monomials*), utilizand interfata Comparable;
* ***anulation(): void*** : metoda care sterge din lista principala(*monomials*), elementul din coada listei secundare(*lastMonomials*), adica ultimul monom adaugat;
* ***mulWithMonom(Monomial):* *Polynomial*:** metoda care inmulteste polinomul curent cu un monom, returnand un alt obiect de tip Polynomial;
* ***substract(Polynomial):* *Polynomial***: metoda care face diferenta dintre doua polinoame si returneaza un nou Polynomial.
* ***multiply(Polynomial):*** Polynomial: metoda care face inmultirea dintre doua polinoame si returneaza un al obiect de tipul Polynomial.
* ***Integrate(): Polynomial***: Metoda care realizeaza operatia de integrare asupra unui polinom, returnand un alt polinom;
* ***Derivate(): Polynomial***: Metoda care realizeaza operatie de derivare a unui polinom, returnand un nou obiect de tipul Polynomial.
* ***Sum(): Polynomial*:** Metoda care realizeaza operatia de aduare a doua polinoame, returnand un nou obiect de tipul Polynomial;
* ***Divide(Polynomial): String***: Metoda care realizeaza impartirea a doua polinoame, mai exact, polinomul actual este impratit la polinomul dat ca parametru. Algoritmul de impartire este asemanator cu impartirea normala, realizata pe numere reale. Aceasta metoda apeleaza metodele mulWithMonomial(Monomial) si substract(Polynomial) pentru buna functionare a algoritmului, iar in mod normal ar trebui sa returneze 2 obiecte de tip Polynomial, una pentru cat si una pentru rest. Pentru a nu crea liste suplimentare, am decis sa returnez un string creat prin printarea polinomului Cat, impreuna cu printarea polinomului Rest, utilizand functia printPolynomial(). Stringul returnat o sa fie incarcat direct in TextFieldul corespunzator rezultatului din Frame-ul principal.
* ***printPolynomial(): String***: Metoda care returneaza un obiect de tipul String care o sa contina polinomul curent. De mentionat este faptul ca, in acest program, coeficientii monoamelor sunt de tipul float, adica de tipul real. In aceasta metoda, inainte ca metoda sa returneze stringul ce va contine polinomul curent, am realizat o conversie a variabilelor din float, in int, acolo unde variabilele de tip float nu au o parte zecimala(toate cifrele de dupa virgula sunt 0). Astfel am mentinut un aspect placut al aplicatiei. In schimb, metoda poate parea greu de inteles si de urmarit, deoarece am luat in cosiderare fiecare caz posibil, ceea ce a cauzat aparitia unui numar considerabil de clauze if-else.
* **Clasa MainController:**

In aceasta clasa am efectuat toate operatiile, dar am si gestionat elementele ce tin de interfata grafica. Pentru a putea efectua acest lucru am creat obiecte de tipul interfatei grafice(PolynomialProcessingView), dar si de tipul Polynomial, iar mai apoi am creat cate o actiune pentru fiecare buton disponibil in interfata grafica, urmand urmatorul algoritm:

* + - 1. Obtine rezultul operatiei dorite, utilizand metode create in clasa Polynomial. Acest rezultat va fi stocat intr-un obiect intermediar de tipul Polynomial, sau de tipul String(in cazul impartirii).
      2. Rezultatul va fi incarcat in JtextFieldul din interfata grafica corespunzator rezultatului, utilizand metoda Polynomial.printPolynomial().
      3. Componentele din frame-ul curent vor fi revalidate, iar mai apoi frame-ul va fi „redesenat”, utilizand comenzile Jframe.revalidate() si Jframe.repain().

Am luat in considerare, iar mai apoi am gestionat, orice exceptie ce ar putea aparea pe parcursul unei executii a programului. Astfel, utilizatorul nu poate adauga date invalide(litere sau caractere speciale), nu poate adauga monoame care nu au un coeficient, iar operatiile in cazul in care unul sau ambele polinoame sunt 0 nu vor genera erori neasteptate.

* **Interfata Grafica**

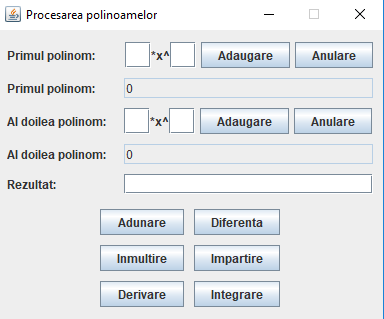


Fig. 1. Interfata grafica

Interfata programului arata ca in Fig. 1. Aceasta este formata dintr-un Jpanel mare, al Frame-ului, in care au fost adaugate succesiv alte panel-uri mai mici. In total am adaugat 5 panel-uri(6 cu Containerul), fiecare fiind atribuit unui rand din interfata(exceptie facand Panelul pentru butoanele cu operatii), avand deasemenea un Layout Manager distinct. Am ales sa merg pe o astfel de abordare, pentru a putea aranaja mai placut componentele, din punct de vedere vizual , fiiecare Layout Manager avand beneficiile si dezavantajele sale, dar si pentru a aprofunda putin acest subiect si pentru a intelege modul de functionare a acestora. Astfel, in interfata mea grafica am folosit urmatorii Layout Manageri: BoxLayout, orientat pe axa X sau Y( in functie de nevoi), pentru Panelul mare si pentru panel-urile care contin componentele aferente polinoamelor; GridBagLayout, pentru panelul cu butoanele operatiilor.

In metoda createGUI() stabilim containerul frame-ului, titlul, layout managerul si locatia de deschidere a acestuia, iar apoi apelam metoda de de aranjare a componentelor. Acesta metoda(arrangeComponents()), in care initializam componentele grafice, stabilim layout managerii pentru fiecar panel in parte, stabilim locatia componentelor si le adaugam la panel-urile corespunzatoare.

Pentru o utlizarea mai usoara a aplicatiei, si pentru a-i oferii o utilitate mai mare, la apasarea unui click pe TextFieldul corespunzator rezultatului, continutul acestui field va fi copiat in Clipboard(asemanator comandei: CTRL+A, CTRL+C), utilizatorul putand copia imediat valoarea rezultatului in orice alt loc doreste.

1. **Testare**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metoda | Date intrare | Rezultat asteptat | Rezultat efectiv | Pass/Fail |
| Sum(Polynomial) | P1: 2x^3 + x^2 +3  P2: 2x^2 + 3x +2 | Adunare: +2\*x^3 +3\*x^2 +3\*x^1 +5 | Adunare: +2\*x^3 +3\*x^2 +3\*x^1 +5 | Pass |
| Sum(Polynomial) | P1: 2x^3 + x^2 +3  P2: 0 | Adunare: +2\*x^3 +1\*x^2 +3 | Adunare: +2\*x^3 +1\*x^2 +3 | Pass |
| Substract(Polynomial) | P1: 2x^3 + x^2 +3  P2: 2x^2 + 3x +2 | Scadere: +2\*x^3-1\*x^2-3\*x^1 +1 | Scadere: +2\*x^3-1\*x^2-3\*x^1 +1 | Pass |
| Derivative() | P1: 2x^3 + x^2 +3 | Derivare: +6\*x^2 +2\*x^1 | Derivare: +6\*x^2 +2\*x^1 | Pass |
| Integrate() | P1: 2x^3 + x^2 +3 | Integrare: +0.500 \*x^4 +0.333 \*x^3 +3\*x^1 | Integrare: +0.500 \*x^4 +0.333 \*x^3 +3\*x^1 | Pass |
| Multiply(Polynomial) | P1: 2x^3 + x^2 +3  P2: 2x^2 + 3x +2 | Inmultire: +4\*x^5 +8\*x^4 +7\*x^3 +8\*x^2 +9\*x^1 +6 | Inmultire: +4\*x^5 +8\*x^4 +7\*x^3 +8\*x^2 +9\*x^1 +6 | Pass |
| Multiply(Polynomial) | P1: 2x^3 + x^2 +3  P2: 0 | Inmultire: 0 | Inmultire: 0 | Pass |
| Divide(Polyomial) | P1: 2x^3 + x^2 +3  P2: 2x^2 + 3x +2 | Catul: +1\*x^1-1 | Restul: +1\*x^1 +5 | Catul: +1\*x^1-1 | Restul: +1\*x^1 +5 | Pass |
| Divide(Polynomial) | P1: 2x^3 + x^2 +3  P2: 0 | Impartitorul trebuie sa fie diferit de 0! | Impartitorul trebuie sa fie diferit de 0! | Pass |

1. **Rezultate**

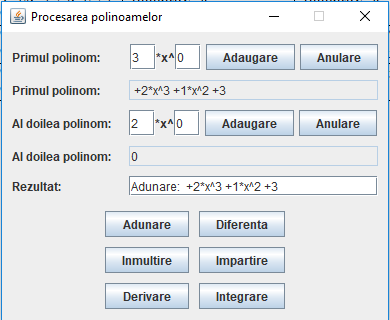


Fig.2.

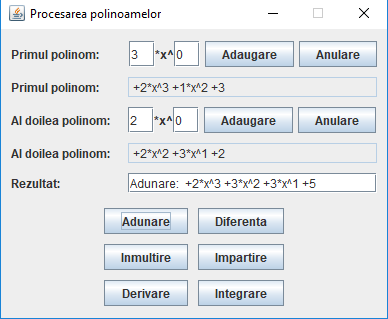


Fig. 3

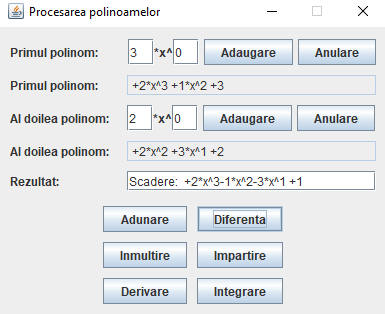


Fig. 4

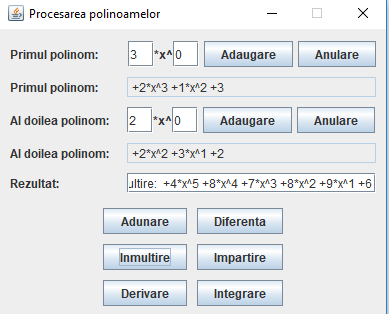


Fig. 5

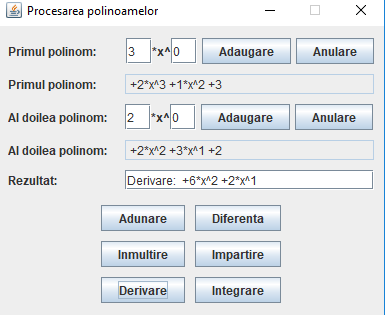


Fig. 6

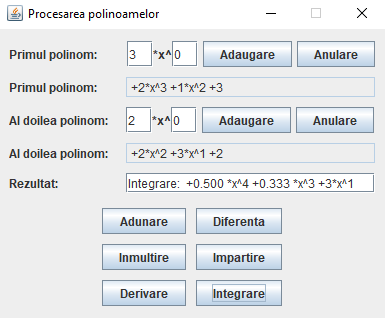


Fig. 7

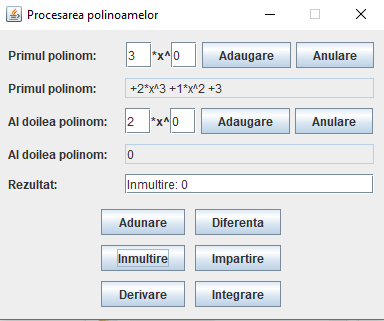


Fig. 8

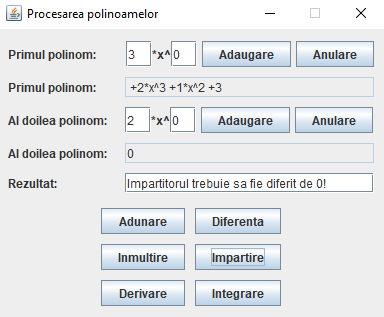


Fig. 9

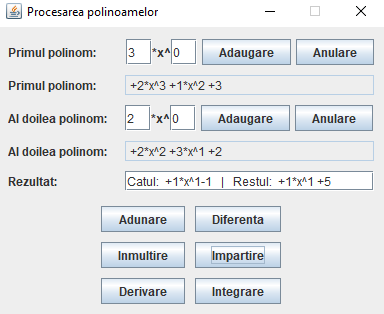


Fig. 10

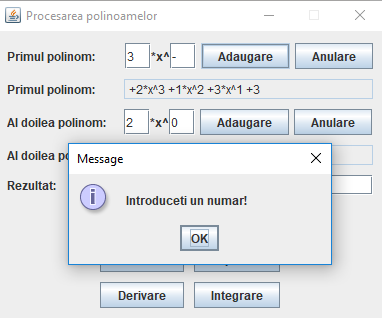


Fig. 11

1. **Concluzii**

In aceasta tema am aprofundat anumite aspecte ale ArrayList-urilor, am invatat sa folosesc metoda de iteratie *for each,* in locul „clasicului” *for*, si am asimilat o buna parte din layout mangerii uzuali folosindu-i pe parcursul proiectului, usurandu-mi astfel, intr-o masura mai mare sau mai mica, crearea urmatoarelor interfete grafice. In schimb, consider ca cel mai important lucru asimilat pe parcursul acestei teme a utilizarea eficienta a arhitecutirii MVC (Model-View-Controller).

7.1. Dezvoltari ulterioare:

* Adaugarea unor operatii suplimentare:
  + aflarea radacinilor
  + Realizarea unor anumite simulari cu rezultat grafic pe baza unor functii polinomiale
  + Utilizarea unor alte biblioteci, cum ar fi cea de la Wolfram Alpha, pentru a putea trimite query-uri cu ecuatii sau grafice mai complicate, sau poate chiar pentru obtinerea de informatii in legatura cu o cerinta a utilizatorului(exemplu: utilizatorul doreste sa afle daca polinomul introdus interpoleaza o functie oarecare; utilizatorul sa aiba posibilitatea sa verifice daca este adevarat sau nu, iar daca nu, sa vada solutia, daca este vreuna).

1. **Bibliografie**

<https://en.wikipedia.org/wiki/Polynomial>

<https://www.geeksforgeeks.org/for-each-loop-in-java/>