Sistem de procesare al polinoamelor

Miholca Cristian-Daniel

Grupa 30223

Îndrumător de laborator: Dorin Moldovan

Cuprins

[1. Cerințe funcționale 4](#_Toc3911725)

[2. Obiective 4](#_Toc3911726)

[2.1. Obiectiv principal 4](#_Toc3911727)

[2.2. Obiective secundare 4](#_Toc3911728)

[3. Analiza problemei 4](#_Toc3911729)

[4. Proiectare 6](#_Toc3911730)

[4.1. Diagrama de clase 6](#_Toc3911731)

[4.2. Clase și algoritmi folosiți 7](#_Toc3911732)

[5. Testare 8](#_Toc3911733)

[6. Concluzii și dezvoltări ulterioare 9](#_Toc3911734)

[7. Bibliografie 9](#_Toc3911735)

# Cerințe funcționale

Dezvoltați un calculator care primește ca date intrare două polinoame și furnizează rezultatul uneia dintre operațiile următoare : adunare, scădere, înmulțire, împărțire, derivare, integrare.

Aplicația trebuie să respecte paradigmele POO.

# Obiective

## Obiectiv principal

Obiectivul principal al proiectului este de a dezvolta un sistem de procesare al polinoamelor de o singură variabilă cu coeficienți întregi.

## Obiective secundare

* Dezvoltarea de use case și scenarii
* Proiectarea claselor
* Alegerea structurilor de date
* Dezvoltarea algoritmilor
* Implementarea soluției
* Testare

# Analiza problemei

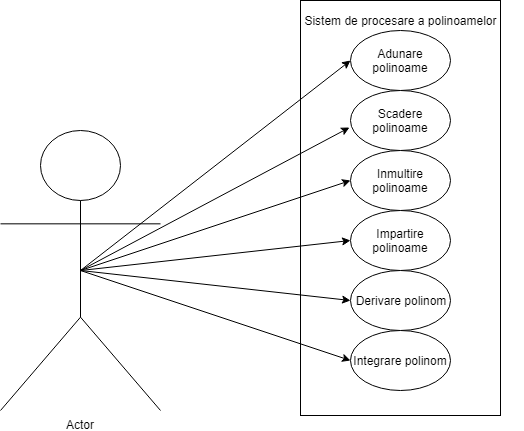
În matematică, un polinom este o expresie construită dintr-una sau mai multe variabile și constante, folosind doar operații de adunare, scădere, înmulțire și ridicare la putere. Spre exemplu, x2+1 este un polinom.

Polinoamele pot fi privite ca o sumă de termeni, numiți monoame, care sunt alcătuiți dintr-un coeficient și o variabilă care este ridicată la putere. În exemplul anterior, x2 este un monom având coeficientul egal cu 1 și exponentul egal cu 2.

Gradul unei variabile într-un monom este egal cu exponentul variabilei din acel monom. Deoarece x=x1, gradul unei variabile fără exponent este unu. Un monom fără variabile se numește monom constant. Gradul unui termen constant este 0.

Algoritmul de utilizare al aplicației este următorul :

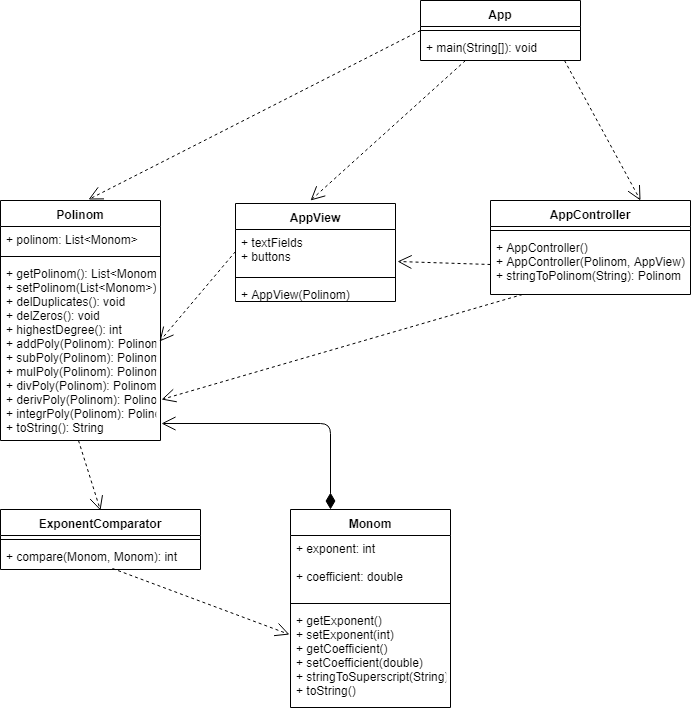
* Utilizatorul introduce două polinoame ;
* Se selectează operația dorită (adunare, scădere, înmulțire, împărțire, derivare sau integrare) ;
* În cazul în care s-a selectat derivare sau integrare, rezultatul va fi calculat pentru primul polinom ;
* Se apasă butonul egal pentru a vizualiza rezultatul ;
* Pentru a șterge datele introduse se apasă butonul clear, iar pentru a reseta aplicația se apasă butonul reset.



# Proiectare

## Diagrama de clase

Diagrama de clase UML pentru aplicație :



Se observă că monomul este parte a polinomului. Celelalte relații sunt relații de dependență de tip "folosește".

## Clase și algoritmi folosiți

În clasa **Monom** am modelat reprezentarea unui monom, adică acesta are ca și câmpuri un exponent întreg și un coeficient de tip double. Am ales ca tipul de dată pentru coeficient să fie double, deoarece în cazul integrării sau împărțirii sunt șanse mari ca acesta să fie un număr real. Funcția stringToSuperscript() are rolul de a scrie exponentul imediat deasupra variabilei. De exemplu : x^2 va fi transformat în x2. Funcția toString() va fi folosită în metoda toString() din clasa Polinom și generează un string corespunzător monomului reprezentat.

**public** String stringToSuperscript(String input) {

**char**[] superscriptDigits = { '\u2070', '\u00b9', '\u00b2', '\u00b3', '\u2074', '\u2075', '\u2076', '\u2077', '\u2078', '\u2079' };

**char**[] text = input.toCharArray();

**for** (**int** i = 0; i < text.length; i++) {

text[i] = superscriptDigits[text[i] - '0'];

}

String rez = String.*copyValueOf*(text);

**return** rez;

}

**public** String toString() {

/\*Stringul returnat\*/

String s = "";

/\*Adaug coeficientul la s\*/

s=s+"+"+**this**.coefficient;

/\*Daca exponentul este 1 nu il afisez\*/

**if** (**this**.exponent == 1) {

s = s + "x";

} **else** {

/\*Afisez exponentul\*/

**if** (**this**.exponent != 0) {

String t = "";

t+=exponent;

s = s + "x" + stringToSuperscript(t);

}

}

**return** s;

}

În clasa **Polinom** am definit operațiile corespunzătoare calculatorului. Polinoamele introduse sunt reprezentate sub forma unor liste de tipul ListArray<Monom>. Astfel, operațiile sunt destul de eficiente, cea mai ineficientă operație fiind cea de înmulțire care rulează în O(n\*m), unde n și m reprezintă numărul de elemente din fiecare listă (sau numărul de monoame). Toate funcțiile, în afară de cea pentru împărțire, returnează un polinom. Operația de împărțire va returna un array de polinoame, de lungime 2, pentru a putea reține câtul și restul.

Pentru operația de **adunare** am ales să parcurg cele două liste și să compar în permanență exponenții monoamelor. Dacă exponenții sunt egali, atunci adun coeficienții, altfel mă deplasez la elementul următor din polinomul în care se găsește monomul cu exponentul mai mare. Algoritmul este asemănător cu algoritmul de interclasare a doi vectori.

**public** Polinom addPoly(Polinom polinom2) {

Polinom result = **new** Polinom();

**this**.polinom.sort(**new** ExponentComparator());

polinom2.polinom.sort(**new** ExponentComparator());

**this**.delDuplicates();

polinom2.delDuplicates();

ListIterator p1 = **this**.polinom.listIterator();

ListIterator p2 = polinom2.polinom.listIterator();

**while** (p1.hasNext() && p2.hasNext()) {

Monom x = (Monom) p1.next();

Monom y = (Monom) p2.next();

**if** (x.getExponent() > y.getExponent()) {

result.polinom.add(x);

p2.previous();

} **else** {

**if** (x.getExponent() < y.getExponent()) {

result.polinom.add(y);

p1.previous();

} **else** {

x.setCoefficient(x.getCoefficient() + y.getCoefficient());

result.polinom.add(x);

}

}

}

**while** (p1.hasNext() || p2.hasNext()) {

**if** (p1.hasNext()) {

result.polinom.add((Monom) p1.next());

}

**if** (p2.hasNext()) {

result.polinom.add((Monom) p2.next());

}

}

result.delZeros();

**return** result;}

Operația de **scădere** este, de fapt, o adunare în care cel de-al doilea polinom este înmulțit cu -1.

**public** Polinom subPoly(Polinom polinom2) {

**int** i;

**for** (i = 0; i < polinom2.polinom.size(); i++) {

**double** oldCoeff = polinom2.polinom.get(i).getCoefficient();

polinom2.polinom.get(i).setCoefficient(oldCoeff \* (-1));

}

Polinom result = **new** Polinom();

result = **this**.addPoly(polinom2);

**return** result;

}

Pentru operația de **înmulțire** am parcurs cele două liste, înmulțind fiecare termen din primul polinom cu fiecare termen din al doilea polinom. La final, am eliminat eventualele duplicate și eventualele valori nule.

**public** Polinom mulPoly(Polinom polinom2) {

Polinom result = **new** Polinom();

**this**.delDuplicates();

polinom2.delDuplicates();

**this**.polinom.sort(**new** ExponentComparator());

**this**.polinom.sort(**new** ExponentComparator());

**for** (Monom x : **this**.polinom) {

**for** (Monom y : polinom2.polinom) {

**double** newCoeff;

**int** newExp;

newCoeff = x.getCoefficient() \* y.getCoefficient();

newExp = x.getExponent() + y.getExponent();

result.polinom.add(**new** Monom(newCoeff, newExp));

}

}

result.delDuplicates();

result.delZeros();

result.polinom.sort(**new** ExponentComparator());

**return** result;

}

Operația de **împărțire** a fost implementată după modelul împărțirii naturale a polinoamelor. Algoritmul este bazat pe teorema împărțirii cu rest.

**public** Polinom[] divPoly(Polinom polinom2) {

Polinom[] result = **new** Polinom[2];

**this**.delDuplicates();

polinom2.delDuplicates();

**if** (**this**.highestDegree() < polinom2.highestDegree()) {

**throw** **new** NumberFormatException("Bad input!");

} **else** {

**this**.polinom.sort(**new** ExponentComparator());

**this**.polinom.sort(**new** ExponentComparator());

Polinom q = **new** Polinom();

Polinom r = **new** Polinom(**this**.polinom);

**while** (r.polinom.isEmpty() == **false** && (r.highestDegree() >= polinom2.highestDegree())) {

**double** newCoef = r.polinom.get(0).getCoefficient() / polinom2.polinom.get(0).getCoefficient();

**int** newPow = r.polinom.get(0).getExponent() - polinom2.polinom.get(0).getExponent();

Polinom t = **new** Polinom();

t.polinom.add(**new** Monom(newCoef, newPow));

q = q.addPoly(t);

r = r.subPoly(t.mulPoly(polinom2));

}

// return qrPair

q.delDuplicates();

r.delDuplicates();

**if**(q.getPolinom().isEmpty()) {

q.getPolinom().add(**new** Monom(0,0));

}

**if**(r.getPolinom().isEmpty()) {

r.getPolinom().add(**new** Monom(0,0));

}

result[0] = q;

result[1] = r;

**return** result;

}

}

Operația de **derivare** se face printr-o singură parcurgere a polinomului. Fiecare monom este derivat în parte după formula: (a\*xn)’=a\*n\*xn-1. Polinomul derivat este cel introdus în caseta de text corespunzătoare polinomului 1.

**public** Polinom derivPoly() {

**this**.delDuplicates();

**this**.polinom.sort(**new** ExponentComparator());

Polinom result = **new** Polinom(**this**.polinom);

List<Monom> deleted = **new** ArrayList<Monom>();

**for** (Monom x : result.polinom) {

**if** (x.getExponent() == 0) {

deleted.add(x);

} **else** {

x.setCoefficient(x.getCoefficient() \* x.getExponent());

x.setExponent(x.getExponent() - 1);

}

}

result.polinom.removeAll(deleted);

**return** result;

}

Operația de **integrare** se face, de asemenea, după formule bine știute. Ca și în cazul derivării, polinomul integrat este polinomul 1.

**public** Polinom integrPoly() {

**this**.delDuplicates();

**this**.polinom.sort(**new** ExponentComparator());

Polinom result = **new** Polinom(**this**.polinom);

**for** (Monom x : result.polinom) {

x.setExponent(x.getExponent() + 1);

**if** (x.getExponent() != 0)

x.setCoefficient(x.getCoefficient() / x.getExponent());

}

**return** result;

}

Clasa **ExponentComparator** reprezintă un comparator folosit pentru a sorta un monoamele în funcție de exponent.

**public** **class** ExponentComparator **implements** Comparator<Monom> {

**public** **int** compare(Monom x, Monom y) {

**if**(x.getExponent()>y.getExponent()) {

**return** -1;

}**else** **if**(x.getExponent()<y.getExponent()) {

**return** 1;

}**else** {

**return** 0;

}

}

}

Aplicația respectă structura MVC: modelul este reprezentat de clasa Polinom, vederea de către clasa AppView, iar controllerul de către clasa AppController.

Astfel, în clasa AppView se specifică cum se prezintă utilizatorului datele modelului. Aplicația conține 3 zone în care poate fi introdus textul, câte una pentru fiecare polinom și una în care este afișat rezultatul. Selecția operației se face cu ajutorul a 6 radio button-uri, dintre care doar unul singur poate fi selectat la un moment dat. Butonul Clear are rolul de a șterge conținutul din fiecare text field. Butonul Reset este pus cu scopul de a reseta aplicația, de a o aduce la condițiile inițiale. La apăsarea butonului „=” se afișează rezultatul operației selectate.

Clasa AppController specifică acțiunile specifice fiecărui buton. De asemenea, conține o funcție, stringToPolinom(String) pe care am folosit-o pentru a converti un string într-un polinom reprezentat sub forma unei liste de monoame.

**public** AppController(Polinom model, AppView view) {

**this**.result = model;

**this**.view = view;

/\*Adaugarea ascultatorilor pentru butoane\*/

view.addResetListener(**new** ResetListener());

view.addClearListener(**new** ClearListener());

view.addEqualsListener(**new** EqualsListener());

view.addDerivateListener(**new** DerivateListener());

view.addIntegrateListener(**new** IntegrateListener());

}

/\*\*Functia converteste un String la tipul Polinom.

\*

\* **@param** input - un string care va fi preluat din textField

\* **@return** result - un polinom

\*/

**public** Polinom stringToPolinom(String input) {

/\*Daca polinom contine 'X' in loc de 'x'\*/

input=input.replaceAll("X", "x");

Polinom result = **new** Polinom();

/\*Despart stringul in mai multe stringuri delimitate de "+" sau "-"\*/

String[] arrOfStrings = input.split("(?=[+-])");

**double** coeff = 0.0;

**int** exp = 0;

/\*Fiecare substring il convertesc intr-un monom\*/

**for** (String s : arrOfStrings) {

/\*Caut indicele caracterului 'x'\*/

**int** indexOfX = s.indexOf('x');

/\*Daca nu exista, inseamna ca monomul este o constanta\*/

**if** (indexOfX == -1) {

coeff = Double.*parseDouble*(s);

exp = 0;

} **else** {

/\*Daca x se afla pe pozitia 0, coeficientul=1\*/

**if**(indexOfX==0) {

coeff = 1;

}

**else** {

String coeffString = s.substring(0, indexOfX);

/\*Daca stringul incepe cu "-x...", coeficientul lui x este -1\*/

**if**(coeffString.compareTo("-")==0)

coeff=-1;

**else** {

/\*Daca stringul incepe cu"+x...",coeficientul este 1\*/

**if**(coeffString.compareTo("+")==0)

coeff=1;

**else**

coeff =Double.*parseDouble*(coeffString);

}

}

String expString = s.substring(indexOfX + 1);

**if** (expString.contains("^")) {

expString = expString.substring(1);

exp = Integer.*parseInt*(expString);

} **else** {

exp = 1;

}

}

result.getPolinom().add(**new** Monom(coeff, exp));

}

**return** result;

}

# Testare

Pentru faza de testare am ales câteva polinoame aleator și le-am testat folosind JUnit.

**public** **class** PolinomTest {

**private** Polinom p1;

**private** Polinom p2;

@Test

**public** **void** testAddPoly() {

p1 = **new** Polinom();

p2 = **new** Polinom();

// p1=x^2 + 1 + 1

p1.getPolinom().add(**new** Monom(1, 2));

p1.getPolinom().add(**new** Monom(1, 0));

p1.getPolinom().add(**new** Monom(1, 0));

// p2=x^3 + x+ 3x^2 + 1 + x^3 + x^3 + 2x^2 + 3 + 7x

p2.getPolinom().add(**new** Monom(1, 3));

p2.getPolinom().add(**new** Monom(1, 1));

p2.getPolinom().add(**new** Monom(3, 2));

p2.getPolinom().add(**new** Monom(1, 0));

p2.getPolinom().add(**new** Monom(1, 3));

p2.getPolinom().add(**new** Monom(1, 3));

p2.getPolinom().add(**new** Monom(2, 2));

p2.getPolinom().add(**new** Monom(3, 0));

p2.getPolinom().add(**new** Monom(7, 1));

Polinom result = **new** Polinom();

result = p1.addPoly(p2);

// result=3x^3 + 6x^2 + 8x + 6

Polinom resTrue = **new** Polinom();

resTrue.getPolinom().add(**new** Monom(3,3));

resTrue.getPolinom().add(**new** Monom(6,2));

resTrue.getPolinom().add(**new** Monom(8,1));

resTrue.getPolinom().add(**new** Monom(6,0));

*assertEquals*(result.toString(),resTrue.toString());

}

@Test

**public** **void** testSubPoly() {

p1 = **new** Polinom();

p2 = **new** Polinom();

// p1=x^2 + 5

p1.getPolinom().add(**new** Monom(1, 2));

p1.getPolinom().add(**new** Monom(5, 0));

// p2=x^3 + x^5 + 2

p2.getPolinom().add(**new** Monom(1, 3));

p2.getPolinom().add(**new** Monom(1, 5));

p2.getPolinom().add(**new** Monom(2, 0));

Polinom result = **new** Polinom();

result = p1.subPoly(p2);

// result=-x^5 - x^3 + x^2 + 3

Polinom resTrue = **new** Polinom();

resTrue.getPolinom().add(**new** Monom(-1,5));

resTrue.getPolinom().add(**new** Monom(-1,3));

resTrue.getPolinom().add(**new** Monom(1,2));

resTrue.getPolinom().add(**new** Monom(3,0));

*assertEquals*(result.toString(),resTrue.toString());

}

@Test

**public** **void** testMulPoly() {

p1 = **new** Polinom();

p2 = **new** Polinom();

// p1=x + 1

p1.getPolinom().add(**new** Monom(1, 1));

p1.getPolinom().add(**new** Monom(1, 0));

// p2=x - 1

p2.getPolinom().add(**new** Monom(1, 1));

p2.getPolinom().add(**new** Monom(-1, 0));

Polinom result = **new** Polinom();

result = p1.mulPoly(p2);

// result=-x^2 - 1

Polinom resTrue = **new** Polinom();

resTrue.getPolinom().add(**new** Monom(1,2));

resTrue.getPolinom().add(**new** Monom(-1,0));

*assertEquals*(result.toString(),resTrue.toString());

}

@Test

**public** **void** testDivPoly() {

p1 = **new** Polinom();

p2 = **new** Polinom();

// p1=x^2 - 1

p1.getPolinom().add(**new** Monom(1, 2));

p1.getPolinom().add(**new** Monom(-1, 0));

// p2=x - 1

p2.getPolinom().add(**new** Monom(1, 1));

p2.getPolinom().add(**new** Monom(-1, 0));

Polinom[] result = **new** Polinom[2];

result = p1.divPoly(p2);

// result=x + 1

Polinom[] resTrue = **new** Polinom[2];

resTrue[0] = **new** Polinom();

resTrue[1] = **new** Polinom();

resTrue[0].getPolinom().add(**new** Monom(1,1));

resTrue[0].getPolinom().add(**new** Monom(1,0));

resTrue[1].getPolinom().add(**new** Monom(0,0));

*assertEquals*(result[0].toString(),resTrue[0].toString());

*assertEquals*(result[1].toString(),resTrue[1].toString());

}

@Test

**public** **void** testDerivPoly() {

p1 = **new** Polinom();

p2 = **new** Polinom();

// p1=x^5 + 3x^2 + 2x +1

p1.getPolinom().add(**new** Monom(1, 5));

p1.getPolinom().add(**new** Monom(3, 2));

p1.getPolinom().add(**new** Monom(2,1));

p1.getPolinom().add(**new** Monom(1,0));

Polinom result = **new** Polinom();

result = p1.derivPoly();

// result=5x^4 + 6x + 2

Polinom resTrue = **new** Polinom();

resTrue.getPolinom().add(**new** Monom(5,4));

resTrue.getPolinom().add(**new** Monom(6,1));

resTrue.getPolinom().add(**new** Monom(2,0));

*assertEquals*(result.toString(),resTrue.toString());

}

@Test

**public** **void** testIntegrPoly() {

p1 = **new** Polinom();

p2 = **new** Polinom();

// p1=3x^2 + 2x + 4

p1.getPolinom().add(**new** Monom(3, 2));

p1.getPolinom().add(**new** Monom(2, 1));

p1.getPolinom().add(**new** Monom(4,0));

Polinom result = **new** Polinom();

result = p1.integrPoly();

// resTrue=x^3 + x^2 + 4x

Polinom resTrue = **new** Polinom();

resTrue.getPolinom().add(**new** Monom(1,3));

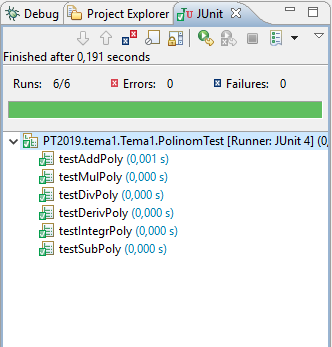
resTrue.getPolinom().add(**new** Monom(1,2));

resTrue.getPolinom().add(**new** Monom(4,1));

*assertEquals*(result.toString(),resTrue.toString());

}

}



# Concluzii și dezvoltări ulterioare

Consider că aplicația realizată de mine este una ușor de folosit, ea putând fi utilizată de către oricine. Interfața este destul de simplă, putându-se realiza orice operație printr-o simplă introducere de două stringuri și apăsarea a două click-uri.

Cred că există multe posibilități dezvoltare ulterioară. Una dintre acestea ar fi tratarea tuturor excepțiilor pentru niște date de intrare care nu respectă formatul folosit de către aplicația mea. O altă dezvoltare posibilă ar fi realizarea mai multor operații sau desenarea graficului pentru polinomul rezultat.

# Bibliografie

* Împărțirea polinoamelor: <https://en.wikipedia.org/wiki/Polynomial_long_division>
* Diagrame: <https://draw.io/>
* Polinom: <https://ro.wikipedia.org/wiki/Polinom>
* Superscript: <https://www.compart.com/en/unicode/block/U+2070>
* JUnit: <https://www.mkyong.com/tutorials/junit-tutorials/>
* Interfata grafica: <https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>
* <https://www.geeksforgeeks.org/arraylist-removeall-method-in-java-with-examples/>