**Tema 1**

Sistem de procesare al polinoamelor

Daniel-Andrei Gherghel

Grupa 30224

Îndrumător de laborator: Claudia Pop

Cuprins

[1. Cerința 3](#_Toc3911725)

[2. Obiective 3](#_Toc3911726)

[2.1. Obiectiv principal 3](#_Toc3911727)

[2.2. Obiective secundare 3](#_Toc3911728)

[3. Analiza problemei 3](#_Toc3911729)

[4. Proiectare 4](#_Toc3911730)

[4.1. Diagrama de clase 4](#_Toc3911731)

[4.2. Clase și algoritmi folosiți 5](#_Toc3911732)

[5. Testare 14](#_Toc3911733)

[6. Concluzii și dezvoltări ulterioare 18](#_Toc3911734)

[7. Bibliografie 18](#_Toc3911735)

# Cerințefuncționale

Dezvoltați un calculator care primește ca date intrare două polinoame și furnizează rezultatul uneia dintre operațiil eurmătoare : adunare, scădere, înmulțire, împărțire, derivare sau 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(reali).

## 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+4x+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, x2este 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.

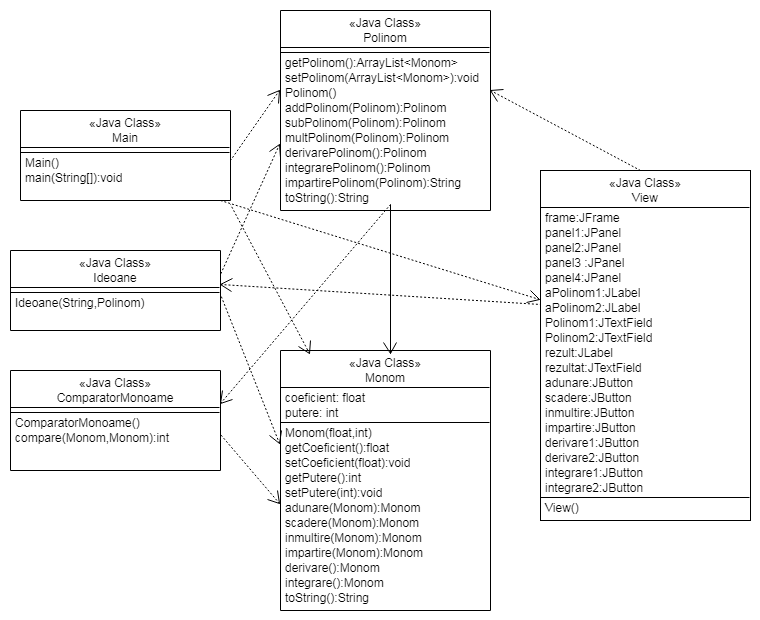
Modul de utilizare al aplicației este următorul :

* Se introduc două polinoame ;
* Se selectează operația dorită (adunare, scădere, înmulțire, împărțire, derivaresauintegrare) ;
* Încazul în care se alege derivare sau integrare, rezultatul va fi calculat pentru primul sau al doilea polinom, în functie de alegerea facuta ;
* La apasarea oricarui buton, rezultatul v-a fi afisat conform operatiei alese;

# Proiectare

## Diagrama de clase

Diagrama UML de clase a aplicației :



## Clase și algoritmi folosiți

**Monom.java**

În clasa **Monom** am modelat un monom, acesta are ca și câmpuri un exponent întreg și un coeficient de tip float. Am ales ca tipul de dată pentru coeficient să fie float pentru a putea lucra cu aceeasi clasă (**Monom**) pentru realizarea operatiilor de impartire si integrare.

Tot în interiorul acestei clase am realizat metodele care efctueză fiecare operatie pe

unul sau două monoame.

Funcția toString() va fi folosită în metoda toString() din clasa **Polinom** avand ca rol generarea un string corespunzător unui monom dat.

**public** **class** Monom {

**protected** **float** coeficient;

**protected** **int** putere;

**public** Monom(**float** coeficient,**int** putere)

{

**this**.coeficient=coeficient;

**this**.putere=putere;

}

**public** Monom adunare(Monom m) {

**return** **new** Monom ( **this**.coeficient+ m.coeficient, putere);

}

**protected** Monom scadere(Monom m) {

**return** **new** Monom( **this**.coeficient - m.coeficient, **this**.putere);

}

**protected** Monom inmultire(Monom m) {

**return** **new** Monom( **this**.coeficient \* m.coeficient, **this**.putere + m.putere);

}

**protected** Monom impartire(Monom m) {

**return** **new** Monom( **this**.coeficient / m.coeficient, **this**.putere - m.putere);

}

**protected** Monom derivare() {

**return** **new** Monom( **this**.coeficient \* **this**.putere, **this**.putere - 1);

}

**protected** Monom integrare() {

**return** **new** Monom( **this**.coeficient / (**this**.putere + 1), **this**.putere + 1);

}

**public** String toString() {

String s="";

**if** (**this**.coeficient > 0) {

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

}

**else** **if** (**this**.coeficient < 0)

{

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

}

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

**if**( **this**.putere ==1)

{

s = s + "x" ;

}

**else** **if**(**this**.putere>1)

{

s = s + "x^" + **this**.putere;

}

}

**return** s;

}

**Polinom.java**

În clasa **Polinom** am definit operațiile corespunzătoare calculatorului utilizand operatiile pe monoame. Polinoamele introduse sunt reprezentate sub forma unor liste de tipul ArrayList<Monom>. Toate funcțiile, în afară de cea pentru împărțire, returnează un polinom. Operația de împărțire va returna un string, care va reține câtul și restul.

Operația de **adunare** am ales să o realizez prin adaugarea tuturor monoamelor care alcătuiesc cele două polinoame. După am comparat terme cu termen pentru a verifica dacă exponenții sunt egali, dacă acestia sunt egali atunci adun și adaug un nou monom in șir ștregand elementele care au fost insumate.

**public** Polinom addPolinom(Polinom p1)

{

Polinom rez=**new** Polinom();

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

rez.polinom.add(aux);

}

**for** (Monom aux : p1.polinom) {

rez.polinom.add(aux);

}

**for** (Monom aux1 : **this**.polinom)

{

**for**(Monom aux2 : p1.polinom)

{

**if** (aux1.getPutere() == aux2.getPutere())

{

rez.polinom.add(aux1.adunare(aux2));

rez.polinom.remove(aux1);

rez.polinom.remove(aux2);

}

}

}

rez.polinom.sort(**new** ComparatorMonoame());

**return** rez;

}

Operația de **scădere** am proiectat-o ca o adunare cu inversul celui de-al doilea polinom .

**public** Polinom subPolinom(Polinom p1)

{

Polinom rez=**new** Polinom();

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

rez.polinom.add(aux);

}

**for** (Monom aux : p1.polinom) {

**float** c=-aux.getCoeficient();

aux.setCoeficient(c);

rez.polinom.add(aux);

}

**for** (Monom aux1 : **this**.polinom)

{

**for**(Monom aux2 : p1.polinom)

{

**if** (aux1.getPutere() == aux2.getPutere())

{

rez.polinom.add(aux1.adunare(aux2));

rez.polinom.remove(aux1);

rez.polinom.remove(aux2);

}

}

}

rez.polinom.sort(**new** ComparatorMonoame());

**return** rez;

}

Pentru operația de **înmulțire** am parcurs cele două polinoame, înmulțind fiecare termen din primul polinom cu fiecare termen din al doilea polinom. La final, am adunat monoamele de acelasi grad si le-am sters din noul polinom .

**public** Polinom multPolinom(Polinom p1)

{

Polinom rez=**new** Polinom();

Polinom rez1=**new** Polinom();

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

rez.polinom.add(aux);

}

**for** (Monom aux : p1.polinom) {

rez.polinom.add(aux);

}

**for** (Monom aux1 : **this**.polinom)

{

**for**(Monom aux2 : p1.polinom)

{

rez.polinom.add(aux1.inmultire(aux2));

rez.polinom.remove(aux1);

rez.polinom.remove(aux2);

}

}

ArrayList<Monom> delete1 = **new** ArrayList<Monom>();

**for** (Monom aux1 : rez.polinom)

{

**for**(Monom aux2 : rez.polinom.subList(rez.polinom.indexOf(aux1)+1, rez.polinom.size()))

{

**if** (aux1.getPutere() == aux2.getPutere())

{

rez1.polinom.add(aux1.adunare(aux2));

delete1.add(aux1);

delete1.add(aux2);

}

}

}

rez.polinom.removeAll(delete1);

**for** (Monom aux : rez.polinom)

{

rez1.polinom.add(aux);

}

rez1.polinom.sort(**new** ComparatorMonoame());

**return** rez1;

}

Operația de **derivare** am realizat-o printr-o singură parcurgere a polinomului. Fiecare monom este derivat în parte după formulă, utilizand operația din cadrul clasei Monom . Polinomul derivat este cel introdus în una din campurile de text corespunzătoare polinomului 1 sau 2 .

**public** Polinom derivarePolinom()

{

Polinom rez=**new** Polinom();

**for** (Monom aux1 : **this**.polinom)

{

rez.polinom.add(aux1.derivare());

}

**return** rez;

}

Operația de **integrare** se face, de asemenea, după aceasi idee . Ca și în cazul derivării, polinomul integrat este polinomul 1 sau 2.

**public** Polinom integrarePolinom()

{

Polinom rez=**new** Polinom();

**for** (Monom aux1 : **this**.polinom)

{

rez.polinom.add(aux1.integrare());

}

**return** rez;

}

Operația de **împărțire** a fost implementată după modelul teoremei împărțirii cu rest. In primă fază am realizat comparea gradelor maxime a celor două polinoame. In cazul în care gradul primului este mai mic decat celui de-al doilea rezultatul este 0.

Dacă se poate realiza **î**mpartirea atunci se incepe construirea polinomului care va reprezenta catul, polinom care va respecta urmatoarea regula:(p1:p2) cat\*p2+restul=p1.

**if** (gradp1<gradp2)

s=s+0;

**else**

**while**(i>=0)

{

Polinom interm=**new** Polinom();

Polinom interm1=**new** Polinom();

String c="";

String v="";

**float** a=aux.polinom.get(j).coeficient/p.polinom.get(0).coeficient;

**float** b=aux.polinom.get(j).putere-p.polinom.get(0).putere;

interm.getPolinom().add(**new** Monom(a,(**int**) b));

Monom m=**new** Monom(a,(**int**) b);

s=s+m.toString();

interm1=p.multPolinom(interm);

aux=aux.subPolinom(interm1);

gradp1--;

i--;

j++;

}

**Ideoane.java**

Constructorul acestei clase are rolul de a transforma un string dat ca intrare,sub forma unui polinom, in stringuri de dimensiunea unui monom. Aceste stringuri vor fi prelucrate si for fi transformate in monoame, monoame care vor alcatui polinomul echivalent dat ca si sir de intrare.

**public** Ideone(String s,Polinom p)

{

Pattern pattern=Pattern.*compile*("([+-]?[^-+]+)");

Matcher matcher = pattern.matcher(s);

**int** x=0;

**int** coeficient,putere;

**while**(matcher.find())

{

x=x+1;

String c=matcher.group(1);

coeficient=Integer.*parseInt*(c.substring(0,2));

**if**(c.length()>2) {

putere=Integer.*parseInt*(c.substring(c.length()-1));

}

**else** {

putere=Integer.*parseInt*(c);

}

p.getPolinom().add(**new** Monom(coeficient,putere));

}

}

**ComparatorMonoame.java**

Clasa **ComparatorMonoame** reprezintă un comparator de monoame folosit pentru a ordona un polinom în funcție de exponentul fiecarui monom care alcatuieste polinomul.

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

@Override

**public** **int** compare(Monom m1, Monom m2) {

// **TODO** Auto-generated method stub

**int** x;

**if**( m1.getPutere() < m2.getPutere() )

x=1;

**else** **if** ( m1.getPutere() > m2.getPutere() )

x=-1;

**else**

x=0;

**return** x;

}

}

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

Aplicatia contine doua campuri in care pot fi introduse datele de intreare, adica cele doua polinoame sub forma unor siruri demonoame de forma +/-ax^b.

Selectia operatiei dorite se face cu ajutorul unuia din cele opt butoane, la un moment dat putand fi selectat un singur buton.

Pentru operatiile de derivare si integrare exista cate doua butoane, putandu-se efectua operatia asupra oricarui polinom introdus de utilizator.

In functie de operatia selectata, pe campul „Rezultat” va fi afisat un string care va reprezenta rezultatul operatiei alese, stingul fiind de aceasi forma : +/-ax^b .

Controlul fiecarui buton a fost implementat tot in cadrul clasei View.

**private** JLabel aPolinom1 = **new** JLabel("Polinom 1: ");

**private** JLabel aPolinom2 = **new** JLabel("Polinom 2: ");

**private** JTextField Polinom1 = **new** JTextField(25);

**private** JTextField Polinom2 = **new** JTextField(25);

**private** JLabel rezult = **new** JLabel("Rezultatul: ");

**private** JTextField rezultat = **new** JTextField(25);

**private** JButton adunare = **new** JButton("Adunare");

**private** JButton scadere = **new** JButton("Scadere");

**private** JButton inmultire = **new** JButton("Inmultire");

**private** JButton impartire = **new** JButton("Impartire");

**private** JButton derivare1 = **new** JButton("Derivare Polinom1");

**private** JButton derivare2 = **new** JButton("Derivare Polinom2");

**private** JButton integrare1 = **new** JButton("Integrare Polinom1");

**private** JButton integrare2 = **new** JButton("Integrare Polinom2");

adunare.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

String s1=Polinom1.getText();

String s2=Polinom2.getText();

Polinom p1=**new** Polinom();

Polinom p2=**new** Polinom();

Ideone i1=**new** Ideone(s1,p1);

Ideone i2=**new** Ideone(s2,p2);

rezultat.setText(p1.addPolinom(p2).toString());

}

});

scadere.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

String s1=Polinom1.getText();

String s2=Polinom2.getText();

Polinom p1=**new** Polinom();

Polinom p2=**new** Polinom();

Ideone i1=**new** Ideone(s1,p1);

Ideone i2=**new** Ideone(s2,p2);

rezultat.setText(p1.subPolinom(p2).toString());

}

});

inmultire.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

String s1=Polinom1.getText();

String s2=Polinom2.getText();

Polinom p1=**new** Polinom();

Polinom p2=**new** Polinom();

Ideone i1=**new** Ideone(s1,p1);

Ideone i2=**new** Ideone(s2,p2);

rezultat.setText(p1.multPolinom(p2).toString());

}

});

derivare1.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

String s1=Polinom1.getText();

Polinom p1=**new** Polinom();

Ideone i1=**new** Ideone(s1,p1);

rezultat.setText(p1.derivarePolinom().toString());

}

});

derivare2.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

String s2=Polinom2.getText();

Polinom p2=**new** Polinom();

Ideone i2=**new** Ideone(s2,p2);

rezultat.setText(p2.derivarePolinom().toString());

}

});

integrare1.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

String s1=Polinom1.getText();

Polinom p1=**new** Polinom();

Ideone i1=**new** Ideone(s1,p1);

rezultat.setText((p1.integrarePolinom()).toString());

// setRez(p1.toString());

}

});

integrare2.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

String s2=Polinom2.getText();

Polinom p2=**new** Polinom();

Ideone i2=**new** Ideone(s2,p2);

rezultat.setText((p2.integrarePolinom()).toString());

// setRez(p1.toString());

}

});

impartire.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

String s1=Polinom1.getText();

String s2=Polinom2.getText();

Polinom p1=**new** Polinom();

Polinom p2=**new** Polinom();

Ideone i1=**new** Ideone(s1,p1);

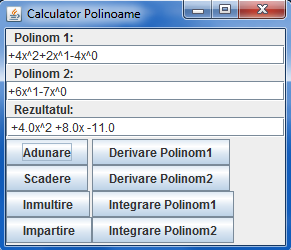
Ideone i2=**new** Ideone(s2,p2);

String s3=p1.impartirePolinom(p2);

rezultat.setText(s3);

}

});



# Testare

Testarea am realizat-o folosind JUnit, introducand polinoame oarecare.

Testarea adunarii :

@Test

**public** **void** AddTest() {

p1 = **new** Polinom();

p2 = **new** Polinom();

// p1=2x^4+2x^3-3x^1+2x^0

p1.getPolinom().add(**new** Monom(2, 4));

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

p1.getPolinom().add(**new** Monom(-3, 1));

p1.getPolinom().add(**new** Monom(2, 0));

// p2=1x^3+x+3x^2+4x^0

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(4, 0));

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

result = p1.addPolinom(p2);

// result=2x^4 + 3x^3 + 3x^2 - 2x + 6

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

resTrue.getPolinom().add(**new** Monom(2,4));

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

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

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

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

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

}

Testarea scaderii :

@Test

**public** **void** SubTest() {

p1 = **new** Polinom();

p2 = **new** Polinom();

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

p1.getPolinom().add(**new** Monom(2, 4));

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

p1.getPolinom().add(**new** Monom(-3, 1));

p1.getPolinom().add(**new** Monom(2, 0));

// p2=x^3 + x+ 3x^2 + 4

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(4, 0));

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

result = p1.subPolinom(p2);

// result=2x^4 + x^3 - 3x^2 - 4x - 2

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

resTrue.getPolinom().add(**new** Monom(2,4));

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

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

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

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

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

}

Testarea inmultirii :

@Test

**public** **void** MultTest() {

p1 = **new** Polinom();

p2 = **new** Polinom();

// p1=-3x + 2

p1.getPolinom().add(**new** Monom(-3, 1));

p1.getPolinom().add(**new** Monom(2, 0));

// p2=3x^2 + 4

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

p2.getPolinom().add(**new** Monom(4, 0));

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

result = p1.multPolinom(p2);

// result= -9x^3 + 6x^2 - 12x + 8

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

resTrue.getPolinom().add(**new** Monom(-9,3));

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

resTrue.getPolinom().add(**new** Monom(-12,1));

resTrue.getPolinom().add(**new** Monom(8,0));

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

}

Testarea impartirii :

@Test

**public** **void** ImpTest() {

p1 = **new** Polinom();

p2 = **new** Polinom();

// p1=2x - 2

p2.getPolinom().add(**new** Monom(2, 1));

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

// p2=3x^2 + 6x + 8

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

p1.getPolinom().add(**new** Monom(6, 1));

p1.getPolinom().add(**new** Monom(8, 0));

String result ="";

result = p1.impartirePolinom(p2);

String resTrue="Catul: +1.5x +3.75 Restul: +11.75";

*assertEquals*(result,resTrue);

}

Testarea derivarii :

@Test

**public** **void** DerivareTest() {

p1 = **new** Polinom();

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

p1.getPolinom().add(**new** Monom(2, 4));

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

p1.getPolinom().add(**new** Monom(-3, 1));

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

result = p1.derivarePolinom();

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

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

resTrue.getPolinom().add(**new** Monom(8,3));

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

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

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

}

Testarea integrarii :

@Test

**public** **void** IntegrareTest() {

p1 = **new** Polinom();

// p1=5x^4 + 8x^3 - 4x + 2

p1.getPolinom().add(**new** Monom(5, 4));

p1.getPolinom().add(**new** Monom(8, 3));

p1.getPolinom().add(**new** Monom(-4, 1));

p1.getPolinom().add(**new** Monom(2, 0));

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

result = p1.integrarePolinom();

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

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

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

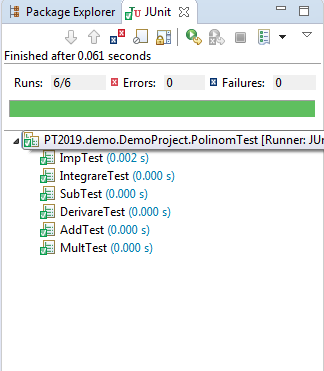
resTrue.getPolinom().add(**new** Monom(2,4));

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

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

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

}



# Concluzii și dezvoltări ulterioare

Dupa parerea mea, aplicatia pe care am creat-o este una destul de simpla, usor de inteles si de asemenea de utilizat. Este o aplicatie eficienta deoarece poate realiza operatii cu polinoame de grad mare in intervale de timp de ordinul secundelor, oferind astfel utilizatorului rezultate rapide si corecte.

Consider ca exista numeroase posibilitati de dezvoltare ulterioara a aplicatiei, spre exemplu, poate fi foarte usor integrata intr-o alta aplicatie care realizeara alte operatii cu polinoame sau o aplicatie care realizeaza reprezentarea grafica a polinoamelor.

# Bibliografie

* <https://en.wikipedia.org/wiki/Polynomial_long_division>
* http://www.umlet.com/umletino/umletino.html?fbclid=IwAR0omFycB9T0LjLKZwmUQvFVwHeuWkp05-fYtLzWNOjreNtaGp7WxjVZtCg
* <https://www.geeksforgeeks.org/arraylist-removeall-method-in-java-with-examples/>
* <https://ro.wikipedia.org/wiki/Polinom>
* https://stackoverflow.com/questions/36490757/regex-for-polynomial-expression
* <https://www.mkyong.com/tutorials/junit-tutorials/>