

**FACULTATEA: Automatică şi Calculatoare**

**SPECIALIZAREA: Calculatoare şi tehnologia informaţiei**

**DISCIPLINA: Tehnici de Programare**

**CALCULATOR POLINOAME**

**Îndrumător Laborator: Realizator:**

Cristina Bianca Pop Oneţiu Alexandru Lucian

Grupa 30228

***Cuprins***

1.Cerinte Functionale

2. Analiza Problemei

3. Obiective

3.1 Obiectivul Principal

3.2 Obiective Secundare

4. Proiectare

4.1 Scenarii si cazuri de utilizare

4.2 Modelare

4.3 Diagrama de clase

5. Implementare

6. Concluzii si Dezvoltari Ulterioare

7. Bibliografie

# Cerinte Functionale

**Cerinta:**

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

**Consideratii generale**

Polinoamele și operațiile cu acestea sunt unele dintre cele mai comune structuri matematice din Computer Science. Motivul principal este simplitatea calcului algebric si analitic a operațiilor, vorbim de operatii aritmetice si operatii de calcul integral. Avantajul este că polinoamele pot aproxima oricât de bine orice funcție continuă și derivabilă, deci pot “interpola” funcții mult mai complexe.

Tocmai de aceea, în multe aplicații se pot regăsi polinoame, într-o formă sau alta, alături de un set de funcții de bază cu care se vor pot manipula.

Scopul acestui proiect este de a implementa operații și funcții de bază pe polinoame, cu scop demonstrativ, pentru a reliefa modul în care acestea pot fi create și manipulate prin intermediul paradigmelor programării orientate pe obiecte. De aceea, materializarea acestui obiectiv va fi reprezentată de crearea unei aplicații de sine stătătoare, independentă, care să poată realiza operații elementare pe un spațiu cât mai larg.

Printre operațiile de bază se vor număra cele de adunare, scădere, înmulțire și împărțire, deivare, integrare dar și o serie de operații auxiliare care nu sunt în directă legătura cu funcționalitățile clasei principale, cum ar fi parsatoare de stringuri.

# Analiza Problemei

Prin analiza problemei, ne referim la un prim set abstract de operații și proprietăți prin care încercăm să depistăm eventualele însușiri și comportamente ale proceselor necunoscute. Programarea orientată ne oferă aici un avantaj clar, tocmai fiindcă ea permite să taclăm problema de la un nivel superiror, fără a mai fi constrâși, într-o așa măsură, de caracteristicile tehnice.

Această strategie de conceptualizare, mai poartă numele și de bottom-up design. Este foarte avantajoasă din prisma găsirii componentelor constituente, deoarece pot fi găsite, relativ ușor, structuri cu o legătură directă în lumea reală( obiecte, acțiuni etc.). Din păcate această versatilitate vine cu prețul complexității, ea crescând spre măsură ce se avansează pe nivelele inferioare.

De cele mai multe ori se pornește de la specificația proiectului, căutându-se:

* Substantive, care devin eventuale clase candidat
* Verbe ce ar putea juca rolul metodelor din clasă.

# Obiective

**2.1** **Obiectivul Principal**

Se doreste crearea unei aplicatii simple Java, care sa realizeze o serie de operatii asupra unor polinome cu coeficienti intregi introduse de utilizator:

* citirea si afisarea unui polinom
* adunarea , scaderea, inmultirea si impartirea a doua polinoame
* operatii care pot fi efectuate asupra unui singur polinom, precum derivarea de ordinal intai si integrarea

Pe langa parte tehnica, din spate, putem gasi si o interfata grafica “ User Friendly ” implementata cu elemente Swing, care poate fi utilizata usor si confortabil de catre orice utilizator.

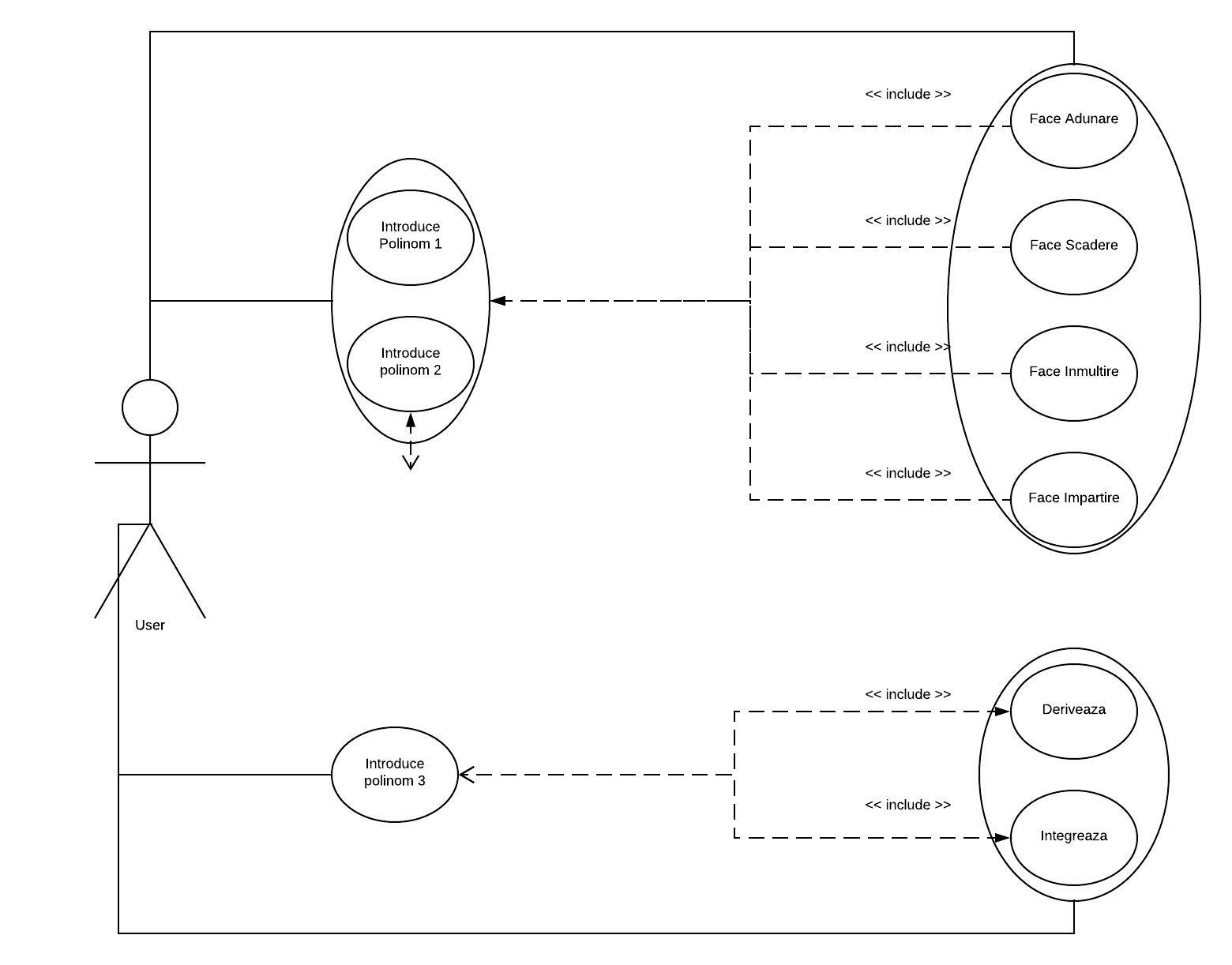
**2.2 Obiective Secundare**

Programul va putea fi accesat de un număr ridicat de persoane, de aceea interfața cu utilizatorul devine punctul de pornire al proiectului. Ea trebuie să permită, într-o manieră cât mai convenabilă, comunicarea utilizatorului cu aplicația.

În cazul Calculatorului de Polinoame, se cunoaște că aplicația trebuie să implementeze operațiile algebrice elementare – or asta implică că trebuie să existe minim două polinoame și cel putin două câmpuri de intrare.

# Obiective

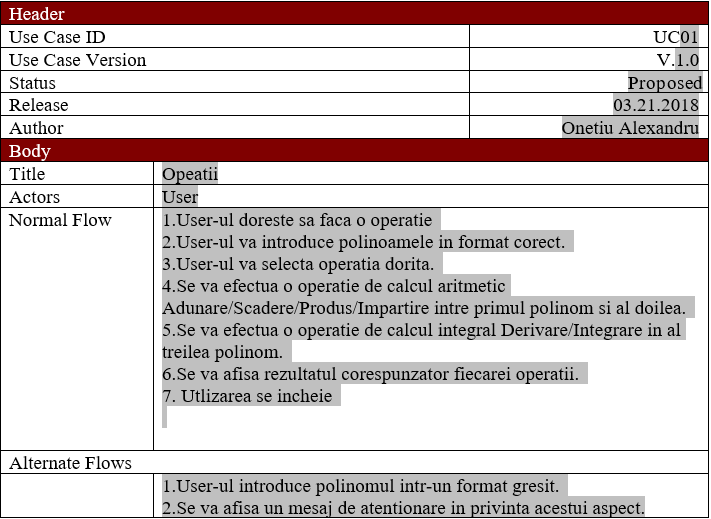
**4.1 Scenarii**



Un user poate introduce polinoame si poate face operatii aritmetice de Adunare/Scadere/Inmultire/Impartire, cat si operatii de calcul integral Deriveaza/Integreaza.

O operatie atat aritmetica cat si de calcul integral are nevoie de polinoame, de unde rezulta relatia de dependenta.

**4.1 Cazuri de utilizare**



**4.2 Modelare**

In procesul de modelare a datelor s-a incercat obtinerea unui program apropiat ca functionalitate si structura de arhitectura MVC ( Model-View-Controller ). Astfel, aplicatia a fost modelata in trei parti separate: modelul , care reprezinta datele aplicatiei, sau Monomul, Polinomul si Operatiile, in partea de Back End si vizualizarea ( view ) regasita in partea care modeleaza interfata cu utilizatorul, deci vizualizarea datelor, si da curs evetualelor operatii si controlorul (controller) care preia intrarile de la utilizator si le transpune in modificari in model, care fac parte din partea de Front End. Concret, pentru rezolvarea cerintei s-au creat 6 clase in 2 pachete diferite.

* **Pachetul BackEnd** :
* **Clasa *Monom :*** utilizata pentru crearea unui monom, format din grad si coeficient. Monomul reprezinta o parte a unui polinom. Mai multe monoame formeaza un polinom.
* **Clasa *Polinom*** : transforma String-ul introdus de la tastatura sub forma unui polinom intr-o lista inlantuita de monoame si il returneaza inpoi ordonat sub forma de String dupa efectuarea de operatii.
* **Clasa Operatie** : va implementa cele 6 operatii..
* **Clasa *CalculatorView*** : realizeaza intreaga interfata cu utilizatorul.
* **Clasa CalculatorController :** asculta la eventualele evenimente produse de utilizator, preia intrarile de la utilizator si afiseaza rezultatul in urma calculelor cu acestea.
* **Clasa *MVCCalculator*** : aici se va afla Main-ul care la rulare va porni aplicatia.

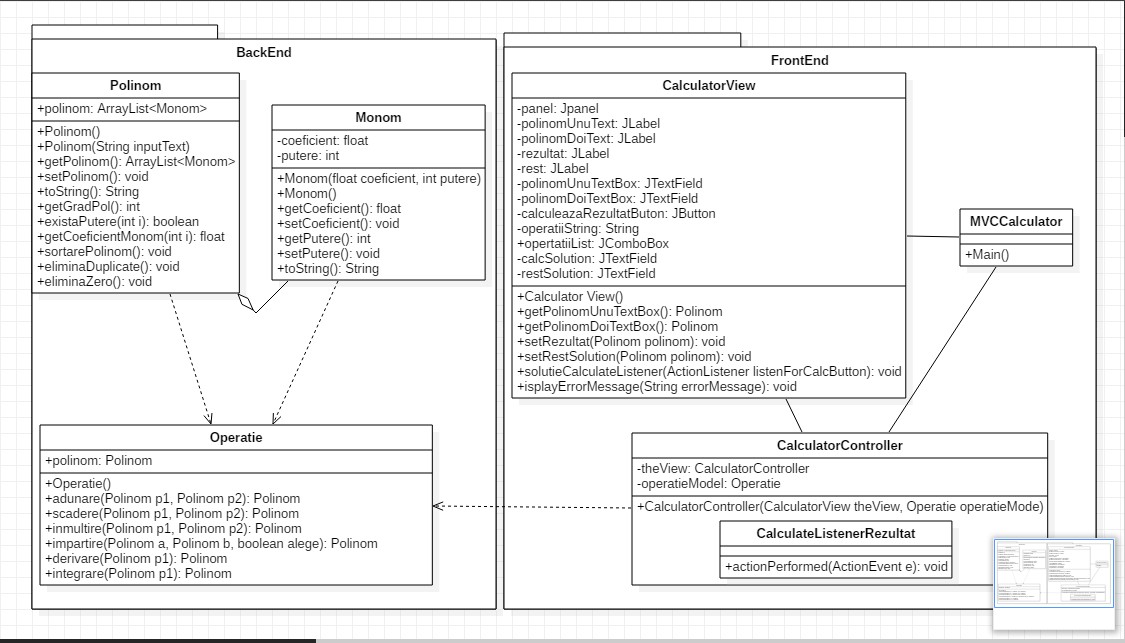
In modelarea datelor, s-a ales in primul rand realizarea clasei Polinom ca un ArrayList de Monoame, la indemnul subtil al cadrelor didactice.

Ulterior am ales separarea clasei Polinom de clasa Operatii, din motive de intretinere ulterioara sau posibila dezvoltare, dar si pentru ca faciliteaza o scriere mai usor de urmarit a codului, continand doar implementarea operatiilor.

Clasa Monom a fost creata pentru a se putea accesa fiecare element al polinomului in mod individual. Aceasta clasa faciliteaza scrierea intregului cod ulterior.

Clasa CalculatorView a fost rezervata pentru codul necesar realizarii interfetei, in timp ce clasa CalculatorController controleaza datele primate si afisate.

**4.3 Diagrama de Clase**



# Implementare

* **Clasa Monom –** aceasta clasa ajuta la formarea unui polinom din mai multe parti mult mai usor de procesat in mod individual : monoame

Atributele:

- coeficient : numar de tip real (float), care reprezinta coeficientul monomului

- grad : numar de tip intreg (int), care reprezinta gradul monomului

Metodele:

- **getPutere()-** returneaza gradul monomului

- **setPutere(int putere)** - primeste ca parametru un grad si fixeaza valoarea variabilei instanta “grad” ca fiind egata cu valoarea primita

- **getCoeficient()-** returneaza coeficientul monomului

-**setCoeficient()-** primeste ca parametru un coeficient si fixeaza valoarea variabilei instanta “coeficient” ca fiind egata cu valoarea primita

-**toString()-** returneza un monom sub forma de string

* **Clasa Polinom** este clasa care modeleaza forma unui polinom si este alcatuita din urmatoarele:

Atribute (variabile instanta):

* Un ArrayList de Monoame – care va contine un ArrayList de monoame, aceastea alcatuind un polinom.

Constructori:

* primul constructor este cel implicit, care nu primeste niciun parametru si nu instantiaza nicio variabila
* al doilea constructor primeste ca si parametru un String str, adica String-ul introdus de la tastatura sub forma unui polinom

Metodele:

* setPolinom(): va avea ca parametru un ArrayList de monoame si pe baza array-ului respectiv va construi polinomul.
* getPolinom(): va returna polinomul
* existaPutere(int i): care pimeste un intreg si va retruna true sau false daca puterea este egala cu acesta.
* getCoefMonom(int i): care in functie parametrul integ primit va returna monomul care este egal cu acesta;
* eliminaDuplicate(): o metoda de tip void care, dupa cum numele sugereaza va efectua operatile necesare eliminarii puterilor duplicate, efectuand opeatia de adunare intre acestea.
* eliminaZero(): metoda care va elimana coeficeintii ce sunt egali cu 0 din polinom;
* sortarePolinom(): metoda ce va sorta polinomul descresccator in functie de putere, folosind bubble sort
* toString():converteste polinomul in srtring usor de citit de catre utilizator;
* **Clasa Operatii** esteclasa care contine metodele pentru operatiile pe polinoame, cat si procesarea rezultatelor operatiilor.

Atribute (variabile instanta):

* nu avem pentru ca ne intereseaza doar metodele

Constructor:

* avem un constructor fara parametri ce va avea o instanta a clasei **Polinom**.

Metodele:

* **adunare (Polinom p1, Polinom p2)**: ia ca argumente 2 obiecte de tip Polinom si returneaza tot un obiect de tip Polinom reprezentand rezultatul adunarii celor 2 date ca argument

Principiu: Sunt parcurse ambele polinoame primite de catre metoda ca parametri cu un ciclu de tip for. Este comparat pe rand cate un monom din primul polinom cu un monom din cel de-al doilea. Se compara gradele: daca acestea sunt egale, se aduna doar coeficientii. Daca insa gradele sunt distincte, se aduna la rezultat monomul din primul polinom.

* **scadere(Polinom p1, Polinom p2)** : ia ca argumente 2 obiecte de tip Polinom si returneaza tot un obiect de tip Polinom reprezentand rezultatul scaderii celor 2 polinoame date ca argument

Principiu: Are acelasi principiu de functionare ca si adunarea, insa coeficientii sunt scazuti in loc sa fie adunati.

* **Inmultire (Polinom p1,Polinom p2):** ia ca argumente 2 obiecte de tip Polinom si returneaza tot un obiect de tip Polinom reprezentand rezultatul inmultirii celor 2 polinoame date ca argument.

Principiu: Cele doua polinoame primite de catre metoda ca si parametri sunt parcurse cu doua cicluri repetitive de tip for. Se inmultesc pe rand cate un monom din fiecare polinom: gradele se aduna, iar coeficientii se inmultesc.

* **Impartire(Polinom p1, Polinom p2,Boolean alege):** ia ca argumente 2 obiecte de tip Polinom si un booleean si returneaza tot un obiect de tip Polinom reprezentand rezultatul impartirii celor 2 polinoame date ca argument, sau restul.

Principiu: Intr-un cicliu repetitiv de tip while,fixam doua monoame care reprezinta monoamele de grad maxim din polinoamele de intrare. Se afla apoi primul monom din cat prin impartirea celor doua monoame de grad maxim: scaderea gradelor si impartirea coeficientilor. Catul se inmulteste cu impartitorul, rezultatul fiind apoi scazut din deimpartit. Procedeul se repeta atata timp cat gradul deimpartitului este mai mare sau egal cu cel al impartitorului.

public Polinom impartire(Polinom a, Polinom b, boolean alege) {

Polinom pol = new Polinom();

Polinom reminder = new Polinom();

Polinom auxPol = new Polinom();

int p1g = a.getGradPol();

int p2g = b.getGradPol();

while (p1g >= p2g) {

auxPol.getPolinom().clear();

int putereDivizer = p1g - p2g;

float coefDivizer = a.getCoefMonom(p1g) / b.getCoefMonom(p2g);

Monom c = new Monom();

c.setCoeficient(coefDivizer);

c.setPutere(putereDivizer);

pol.getPolinom().add(c);

for (Monom monomPo2 : b.getPolinom()) {

Monom m = new Monom();

m.setCoeficient(-(monomPo2.getCoeficient() \* coefDivizer));

m.setPutere(monomPo2.getPutere() + putereDivizer);

auxPol.getPolinom().add(m);

}

Operatie o = new Operatie();

reminder = o.adunare(a, auxPol);

a = reminder;

p1g--;

pol.sortPolinom();

reminder.sortPolinom();

reminder.elimina0();

pol.elimina0();

}

if (alege == true)

return reminder;

else

return pol;

}

* **Derivare (Polinom p1):** primeste un polinom ca argument si returneaza un polinom reprezentand polinomul derivat

Operatia este simplu de implementat deorece pentru realizarea acesteia este necesara inmultirea coeficientului fiecarui monom din polinomul de intrare cu gradul aceluiasi monom, apoi scaderea gradului cu o unitate.

* **Integrare(Polinom p1)**: primeste un polinom ca argument si returneaza un polinom reprezentand valoarea integralei nedefinte din functia reprezentata de polinom

Pentru realizarea acestei operatii sunt necesare cresterea gradului cu o unitate și impartirea coeficientului cu valoarea gradului dupa crestere. Se realizeaza acest procedeu pe fiecare monom din polinomul primit ca parametru.

* **Clasa CalculatorView** este clasa principala a proiectului. La inceputul acesteia se importa elementele necesare interfetei grafice din biblioteca Swing (JPanel, JTextField, Jlabel, Jbutton).

Aceasta clasa extinde clasa Jframe, dar in acelasi timp implementeaza interfata ActionListener.

* **Clasa** **CalculatorController** este clasa care decide ce urmează să facă modelul.Modeleaza elemente din biblioteca awt.event.\* (ActionEvent, ActionListener) necesare ascultarii de evenimente. Adesea, utilizatorul are controlul prin intermediulunei GUI în acest caz, GUI şi Controlorul suntadesea acelaşi.

1. **Concluzii si Dezvoltari Ulterioare**

Din această primă temă, mi-am dat seama, din păcate la final de importanța sepărării, la nivel conceptual, a parsării string-urilor fată de interpretarea lor de către clasele Polinom și Monom. Această greșeală am facut-o în principal, fiindcă nu am reușit să estimez complexitatea reală a problemei. O clasă separată, Parsare, de exemplu, ar fi fost o alternativă mult mai viabilă fiindcă în acest fel nu ar fi fost nevoie să existe vreo legătură între procesul de determinare al monoamelor din string-uri și crearea efectivă a polinomului.

Ca și dezvoltări ulteriore, ideea de a converti dintr-o bază polinomială într-o alta pare foarte atractivă, necesitând algoritmi pentru manipularea matricilor de transformare a bazelor. Desigur, o bază generală ar pune mari probleme de implementare, dar și din punct de vedere matematic, așa că implementarea unor baze standard(Laplace, Newton, Legendre) pare o alternativă fezabilă.

Ce am învățat este rolul pe care separarea obiectivelor îl are chiar la începutul problemei. Nu o dată, a trebuit să elimin bucăți semnificative de cod fiindcă încercam să compensez lipsa de rigurozitate din faza de planificare cu clauze suplimentare.

# Bibliografie

<https://www.tutorialspoint.com/uml/uml_basic_notations.htm>

<https://www.uml-diagrams.org/index-examples.html>

<http://tynerblain.com/blog/2006/01/04/foundation-series-structured-requirements/>

<https://stackoverflow.com/questions/36490757/regex-for-polynomial-expression>

<http://users.utcluj.ro/~igiosan/Resources/POO/Curs/POO09.pdf>

http://www.agiledata.org/essays/objectOrientation101.html

diferite siteuri si tutoariale de pe net