

**Facultatea de Automatică și Calculatoare**

**Disciplina: Tehnici de programare**

**Sistem de procesare a polinoamelor**

**Bud Denisa Maria**

**Grupa 30227**

**An II**

Contents

[1.Obiectivul temei 3](#_Toc477102688)

[2. Analiza 4](#_Toc477102689)

[2.1. Analiza problemei si modelare 4](#_Toc477102690)

[2.2 Scenarii cazuri de utilizare 4](#_Toc477102691)

[3. Proiectare 5](#_Toc477102692)

[3.1 Diagrama UML 5](#_Toc477102693)

[3.2 Structuri de date 7](#_Toc477102694)

[3.3 Clase 7](#_Toc477102695)

[3.4 Algoritmi 8](#_Toc477102696)

[3.5 GUI 9](#_Toc477102697)

[4. Testare 11](#_Toc477102698)

[5. Rezultate 12](#_Toc477102699)

[6. Concluzii 12](#_Toc477102700)

[7. Bilbiografie 12](#_Toc477102701)

# Obiectivul temei

Enuntul temei de laborator este “Propuneti, proiectati si implementati un sistem de procesare a polinoamelor de o singura variabila cu coeficienti intregi.”

Polinoamele și operațiile cu acestea sunt unele dintre cele mai comune structuri matematice din Computer Science data fiind simplitatea algebrică a operațiilor. Polinoamele pot aproxima oricât de bine orice funcție continuă și derivabilă, deci pot “interpola” funcții mult mai complexe.

Obiectivul acestui proiect este de a implementa operațiile 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, derivare și integrare.

În final, voi pune accentul pe afișarea grafică a polinomului prin intermediul unor aproximări liniare cu pas mic, în felul acesta “păcălind” observatorul.

# Analiza

## 2.1. Analiza problemei si modelare

Analizarea problemei presupune depistarea eventualele însușiri și comportamente ale proceselor necunoscute, ceea ce este facilitata de programarea orientata pe obiect, dezvoltata pe ideea ca obiectele pe care dorim sa le manipulam sunt mai importante decat logica necesara pentru manipularea lor.

În [matematică](https://ro.wikipedia.org/wiki/Matematic%C4%83), un polinom este o [expresie](https://ro.wikipedia.org/w/index.php?title=Expresie_matematic%C4%83&action=edit&redlink=1) construită dintr-una sau mai multe [variabile](https://ro.wikipedia.org/wiki/Variabil%C4%83) și [constante](https://ro.wikipedia.org/wiki/Constant%C4%83_matematic%C4%83), folosind doar operații de [adunare](https://ro.wikipedia.org/wiki/Adunare), [scădere](https://ro.wikipedia.org/wiki/Sc%C4%83dere), [înmulțire](https://ro.wikipedia.org/wiki/%C3%8Enmul%C8%9Bire_(matematic%C4%83)) și ridicare la putere constantă.

Polinoamele sunt construite din [termeni](https://ro.wikipedia.org/w/index.php?title=Termen_(matematic%C4%83)&action=edit&redlink=1) numiți [monoame](https://ro.wikipedia.org/w/index.php?title=Monom&action=edit&redlink=1), care sunt alcătuite dintr-o constantă (numită [coeficient](https://ro.wikipedia.org/wiki/Coeficient)) înmulțită o variabilă. Fiecare variabilă poate avea un [exponent](https://ro.wikipedia.org/w/index.php?title=Exponent&action=edit&redlink=1) constant.

În cazul sistemului de procesare a polinoamelor, se cunoaște că aplicația trebuie să implementeze operațiile algebrice elementare, deci aceasta implică că trebuie să existe minim două polinoame. Modul de codificare al informației, dar și formatul acesteia ocupă un rol foarte important.

Metoda aleasa de mine pentru a procesa polinoamele introduse de catre utilizator este reprezentată de citirea pe rand a datelor unui monom: coeficientul si puterea, pentru unul dintre cele 2 polinoame care pot fi retinute, la alegere.

## 2.2 Scenarii cazuri de utilizare

Cazurile care pot fi întâmpinate sunt:

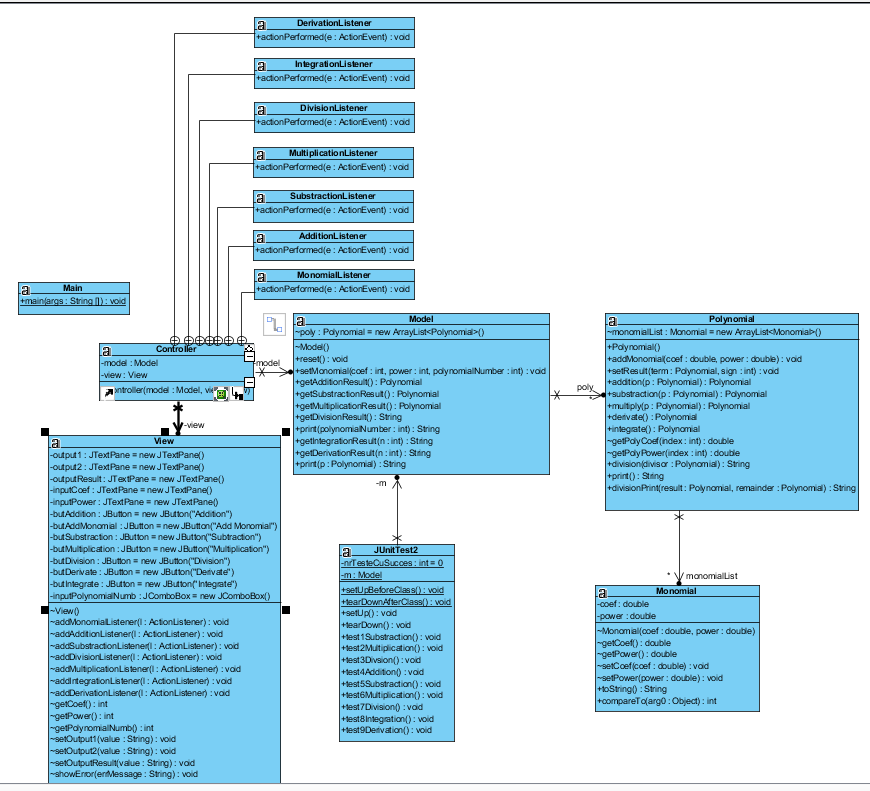
* introducerea unor stringuri în câmpurile coeficient si putere, programul arunca o exceptie si deschide o fereastra care anuntă utilizatorul ca textul este invalid.
* introducerea unui coeficient sau putere reala, reactia programului fiind aceeasi ca cea precedentă;
* introducerea unui numar intreg care depaseste MAX\_INT si MIN\_INT, tratata corespunzator prin fereastra de avertizare;
* calculare suma/ diferența/ produsul/ impărțire între primul polinom și al doilea;
* calculare derivata sau integrala unui polinom ales;

Utilizarea normală a programului presupune introducerea coeficientului si a puterii unui monom, apoi adaugarea acestuia la oricare dintre cele 2 polinoame. O data ce fiecare polinom are cel putin un termen, are sens sa se efectueze operatii cu ele.

# 3. Proiectare

## 3.1 Diagrama UML

Diagrama UML prezintă clasele și metodele prezentate în fiecare clasă.



Proiectarea claselor am realizat-o dupa sablonul Model-View-Controller, datorita izolării logicii programului față de considerentele interfeței cu utilizatorul, rezultând o aplicație unde aspectul vizual și nivelele inferioare ale regulilor sunt mai ușor de modificat, fără a afecta alte nivele.

Intre clasa Polynomial si Monomyal exista o stransa legatura, un polinom contine termeni numiti monoame, relatia dintre cele 2 fiind una de compozitie.

## 3.2 Structuri de date

In programarea orientata pe obiect, legatura stransa dintre date si operatiile efectuate asupra lor se extinde si asupra structurilor de date. Fiecare obiect este o structura de date, asociata cu o colectie de metode prin intermediul carora se manipuleaza aceste date. Obiectele care au aceeasi structura si aceleasi metode se grupeaza intr-o clasa.  Programul este un ansamblu de clase si obiecte care comunica intre ele prin mesaje.

Pe lângă tipurile de date primitive existente în clasele specificate mai sus, se folosesc cu preponderență și colecții. Una din cele mai importante colecții se află în clasa Polinom, ce conține o listă de monoame.

O clasa care implementeaza interfata List este ArrayList, o alternativa excelenta la tablourile unidimensionale. Am utilizat aceasta structura de date in program, in clasa Polynomial ce contina o lista de monoame, deoarece are o utilizare mai buna a memoriei si administrarea elementelor listei este mai facila. Adaugarea elementelor in lista se face foarte usor, prin apelarea metodei .add().

## 3.3 Clase

Clasele folosite sunt:

* **Monomyal** - clasa fundamentală a proiectului ce definește structura de monom, entitate ce conține doar un coeficient de tip real si o putere. Coeficientii si puterile valide vor fi primite sub forma de numar intreg, optiunea de a le salva intr-un tip real se datoreaza existentei operatiilor de impartire si integrare care necesita ca rezultatul sa fie format din valori reale. Pentru accesarea coeficientului si puterii unui monom se pot apela metodele accesoare si mutatoare, variabilele instanta fiind private. Aceasta clasa suprascrie metoda toString a clasei Object pentru a afisa rapid monomul. De asemenea, clasa Monomyal implementeaza interfata Comparable, suprascriind metoda compareTo a carei functionalitate este necesara pentru a sorta descrescator in functie de putere o colectie de date contind monoame.
* **Polynomial** -  permite crearea polinoamelor si adaugarea in colectia de date ArrayList<Monomyal> o serie de monoame, entități și mai fundamentale, dar conține și partea de sortare a acestora. Se tratează cazul in care utilizatorul introduce mai multe monoame de acelasi grad, metoda de adaugare adunand termenii cu aceeasi putere, sau eliminand din lista termenii care au coeficientul 0.

Aceasta metoda de adaugare a unui nou monom in polinom joaca un rol foarte semnificativ, deoarece este folosita si la operatiile de adunare si scadere. Asadar metodele care efectueaza operatiile sunt mult mai compacte si se evita repetitia liniilor de cod.

Polinomul mai contine 2 metode de afisare, cea a polinomului care o apeleaza, fara parametrii si cea care afiseaza catul si restul dintr-o impartire, primite prin 2 parametrii de tip polinom.

* **Model** - Această parte a controlatorului manipulează operațiunile logice și de utilizare de informație (trimisă dinainte de către rangul său superior, in cazul acesta, main) pentru a rezulta de o formă ușor de înțeles. Contine metode mutatoare si accesoare pentru rezultatele obtinute in Polynomial.
* **View** - Acestui membru al familiei îi corespunde reprezentarea grafică, sau mai bine zis, exprimarea ultimei forme a datelor: interfața grafică ce interacționează cu utilizatorul final. Rolul său este de a evidenția informația obținută până ce ea ajunge la controlator.
* **Controlle**r Cu acest element se controleaza accesul la aplicație. În acest fel putem diversifica conținutul nostru de o formă dinamică și statică, în același timp. Constructorul instantiaza 2 variabile de tip model si view, iar pe varibila view sunt adaugati o serie de ascultatori creati tot in Controller, a caror creeare este posibila prin implementarea interfetei ActionListener de catre clasele create care doresc sa ii urmeze comportamentul.
* **Main** care instantiaza modelul, view si controllerul;
* **JUnitTest2** folosita pentru testarea proiectului.

## 3.4 Algoritmi

Cei mai semnificativi algoritmi utilizati in acest proiect sunt cei care urmaresc obtinerea rezultatelor operatiilor pe polinoame din clasa Polynomial, continute in metode public pentru a putea fi accesate din clasa Model.

* **addMonomia**l (double coef, double power) adauga un monom la lista de monoame a polinomului care o apeleaza, avand datele trimise ca parametrii. Lista de monoame este mai intai parcursa cu ajutorul unui iterator pentru a se verifica daca nu exista deja un monom cu acelasi grad, caz in care se aduna coeficientii corespunzatori. Altfel, un nou monom este adaugat la capatul listei. Ordinea gradelor nu conteaza aici, sortarea listei asigurandu-se mereu inainte de afisare.
* **SetResult(Polynomial term, int sign)** este apelata de polinomul rezultat al adunarii si scaderii. Sign este o valoare unitara pozitiva sau negativa, care desemneaza adunare sau scadere (1 sau -1). Metoda parcurge intr-o bucla foreach toate monoamele polinomului term si le adauga la lista de monoame apeland metoda addMonomial cu precizarea ca semnul trimis al coeficientului sa fie in conformitate cu operatia aleasa, adica coeficientul ce se trimite este inmultit cu sign, asa incat din lista se va scadea sau aduna monomul.
* **Addition(Polynomial p)** si **substraction(Polynomial p)** returneaza un polinom care contine rezultatul adunarii, respectiv scaderii polinomului ce apeleaza metoda cu polinomul trimis ca parametru. Ambele executa 2 apeluri de catre polinomul rezultat la metoda setResult() careia ii trimit ca parametru cei 2 operanzi, pe rand, cu variabila sign pozitiva pentru ambii operanzi, in cazul adunarii si in cazul scaderii negativa pentru cel de-al doilea operand.
* **Multiply(Polynomial p)** returneaza un polinom care contine rezultatul inmultirii polinomului ce apeleaza metoda cu polinomul trimis ca parametru. Algoritmul de implementare este simplu, se adauga la lista de monoame a rezultatului cate un nou monom rezultat in 2 foreach-uri imbricate ce parcurg cele 2 polinoame, cunoscand regulile de inmultire a 2 polinoame: inmultirea coeficientilor si adunarea puterilor.
* **Division(Polynomial divisor)** returneaza rezultatul impartirii polinomului ce apeleaza metoda cu polinomul trimis ca parametru. Arunca o exceptie in cazul in care impartirorul este nul; Algoritmul se executa atata timp cat puterea deimpartitului este mai mare sau egala cu cea a impartitorului. Se imparte primul termen al deimpartitului cu primul termen al impartitorului si se obtine primul termen al catului. Termenul astfel obtinut se inmulteste cu impartitorul si se scade din deimpartit si se obtine astfel primul rest al impartirii. Deimpartitului i se atribuie restul si procedeul se reia de la capat cat timp conditia initiala este adevarata.
* **Integrate()** returneaza un polinom egal cu integrala din polinomul care apeleaza metoda. Intr-un foreach se adauga pe rand cate un monom din lista polinomului sursa si se adauga la rezultat monomul cu coeficientul sursa impartit la putere + 1 si puterea incrementata.
* **Derivate()** returneaza un polinom egal cu derivata polinomului care apeleaza metoda. Intr-un foreach se adauga pe rand cate un monom din lista polinomului sursa si se adauga la rezultat monomul cu coeficientul sursa inmultit cu puterea si puterea derementata.

## 3.5 GUI

**GUI = Graphical User Interface**, este legătura dintre utilizator și progam. Folosim această interfață deoarece nu orice persoană este obișnuită cu mediul de lucru Eclipse și are cunoștințe de progamare. Astfel, interfața implementează toate metodele prezentate mai sus prin butoane și TextField, pentru a putea fi utilizate de oricine.

Aceasta a fost implementată prin importarea bibliotecii swing, unde există mai multe elemente care pot fi introduse în interfață.

Interfata o serie de elemente, aflate in clasa View care extinde clasa JFrame.

Frame este locul unde sunt introduse toate elementele, rama, window, etc. Se poate ieși din progam dacă se apasă x în căsuța din dreapta sus (EXIT\_ON\_CLOSE).

**Butoane** = sunt mai multe butoane care pot fi utilizate de utilizator, declarate ca fiind private ( exemplu: private JButton derivare ), fiecare având specifică o operație. Sunt introduse în „ramă” prin frame.add ( numele butonului ).

**TextField** = sunt căsuțe în care se pot introduce taste de la tastatură. De asemenea, pot fi folosite și pentru a afișa rezultatul și să nu se poată introducă datele de la tastura, astfel acestea apar cu o culoare diferită.

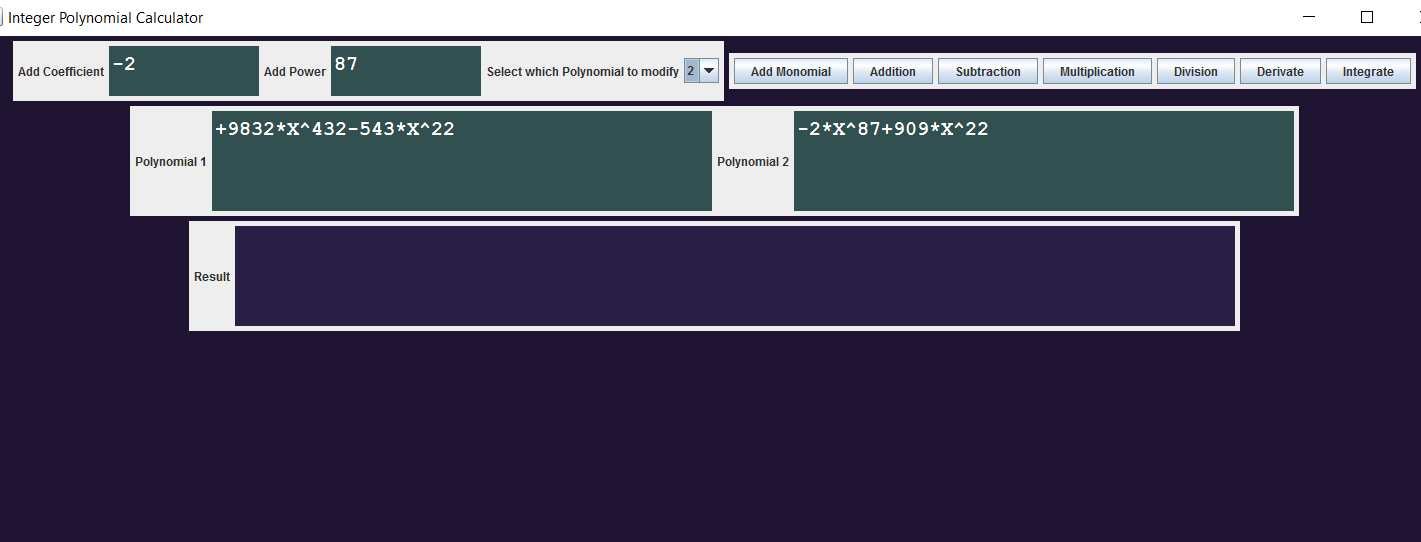
**Label** = este efectiv o etichetă, care poate fi titlul, informații, indicații etc. care pot ajuta utilizatorul să folosească progamul.

Toate aceste elemente au fost introduse în interfață prin frame.add . Elementele de grafică au fost aranjate in containere diferite si adaugate la la containerul care le cuprinde pe toate. Toate elementele acestea sunt întroduse în constructorul public al clasei care se ocupa de interfata, View. Astfel, de câte ori apelăm main-ul, se va face o interfață nouă.

Pentru fiecare buton s-a făcut o metodă nouă care primeste ca parametru un ascultator ActionListener și este adaugat la buton. Astfel, de fiecare dată când are loc o acțiune la un buton, de exemplu a fost apăsat, ascultatorul butonului definit in Controller executa metoda actionPerformed(ActionEvent e).

Interfata este compactata prin faptul ca exista un singur camp de citire a coeficientului, respectiv a puterii, corespondenta cu polinomul caruia ii apartin datele este dat de un ComboBox ce contine cifrele 1 si 2, adica polinomul 1 si polinomul 2. Utilizatorul are posibilitatea sa introduca in orice ordine doreste gradul monomului, programul sortand in mod descendent in functie de putere monoamele si le afișează în timp real pe parcursul introducerii datelor. De asemenea, daca se introduc monoame cu acelasi grad, coeficientii lor se însumează pentru a constitui un singur termen in polinom.

Se pot adăuga noi termeni in cele 2 polinoame alegând în orice ordine carui polinom să ii corespundănoul monom, asadar polinoamele pot fi modificate în orice moment.



5

8

6

7

4

3

2

1

1. Camp de introducere a unei valori intregi reprezentand coeficientul monomului curent - TextField;
2. Camp de introducere a unei valori intregi reprezentand puterea monomului curent - TextField;
3. ComboBox cu valorile 1 si 2 desemnand polinomul in care sa se introducă monomul curent;
4. Adaugare monom la polinomul selectat - button;
5. Operatiile de efectuat pe cele 2 polinoame - button;
6. Afisare polinom1 -TextField;
7. Afisare polinom2 – TextField;
8. Afisare rezultat – TextField needitabil;

# 4. Testare

Elementele de testare unitara, pun la dispozitie unelte pentru a inregistra si repeta teste, pentru ca testele unitare sa poata fi repetate usor mai tarziu (de regula cand se schimba o parte din sistem), astfel incat dezvoltatorul sa fie convins ca noile modificari nu au stricat vechea functionalitate. Acest lucru e cunoscut ca testare regresiva

Pentru testarea acestui proiect, am utilizat framework-ul de testare Junit care include următoarele caracteristici principale :

o Fixturi

o Aserții

o Test suites

o Test runners

o Clase JUnit

Aserțiile sunt metode de bază din clasa Assert utilizate pentru testare. Ordinea parametrilor dintr-o metodă de tip assert este: parametru așteptat, parametru actual. În mod normal un test oarecare eșuează dacă o metodă assert va eșua. Metoda clasei Assert utilizata de mine este assertEquals(primitive expected, primitive actual) care verifică dacă două primitive sunt egale. Aceasta mi-a permis sa compar rezultaul asteptat cu cel obtinut in urma unei operatii pe polinoame. Am alcatuit teste pentru fiecare operatie in parte.

# 5. Rezultate

Procesorul de polinoame pe care l-am implementat efectueaza operatii pe polinome cu coeficienți întregi în unele cazuri returnând polinoame cu coeficienți reali. Procesorul de polinome efectuează următoarele operații:

1. Operații pe câte un polinom:

a. Derivare;

b. Integrare;

2. Operații pe 2 polinome:

a. Adunare;

b. Scădere;

c. Înmulțire ;

d. Împărțire

3 Rezultatul pentru fiecare dintre operații este afișat în interfață.

# 6. Concluzii

Prin acest proiect am reusit sa imi insusesc mai bine paradigmele OOP si sa imi perfectionez modul de structurare a claselor. De asemenea, am aprofundat notiunile legate de arhitectura Model-View-Controller, fiind o metoda eleganta de a organiza o aplicatie.

# 7. Bilbiografie

<http://users.utcluj.ro/~igiosan/teaching_poo.html>

<https://docs.oracle.com/javase/>

<http://stackoverflow.com>

<http://mate123.ro/formule-matematice-algebra-liceu-generala/impartirea-polinoamelor/>

https://google.github.io/styleguide/javaguide.html

http://www.mkyong.com/tutorials/junit-tutorials/

https://www.youtube.com/results?search\_query=java+swing