

Calculator Polinoame

Fariseu Adriana-Teodora

Facultatea de Automatică și Calculatoare, Grupa 30225

Anul 2020-2021

# **Obiectivul temei**

Problema prezentată solicită realizarea unui calculator ce permite efectuarea unor operații cu polinoame, printre care se numără adunarea, scăderea, înmulțirea, împărțirea, derivarea și integrarea, toate acestea selectate dintr-o interfață grafică.

În încercarea de proiectare și implementare a acestor lucruri însă, se constată anumite cerințe suplimentare prin care trebuie să se treacă pentru a se ajunge la produsul final:

|  |  |  |
| --- | --- | --- |
| Obiectiv | Descriere | Capitol în care este detaliat |
| Analiza problemei | Pentru a transpune problema în cod, trebuie mai întâi să se extragă din enunț informații relevante în legătură cu funcționalitățile și cazurile de utilizare. | [2](#_Analiza_problemei) |
| Proiectarea | Odată extrase informațiile necesare, se poate începe maparea acestora în paradigma de programare pusă la dispoziție (Programare Orientată pe Obiect în Java în cazul acesta) | [3](#_Proiectare) |
| Implementarea | În acest pas se realizează scrierea codului pentru realizarea calculatorului de polinoame, pe baza mapării realizate anterior. | [4](#_Implementare) |
| Testarea | Pentru a ne asigura că produsul final sau chiar rezultate parțiale necesare pentru o etapă următoare funcționează corect, codul implementat trebuie la rândul lor supus unor teste ce simulează scenarii de utilizare identificate în etapa de analiză. | [5](#_Testare_și_rezultate) |

# Analiza problemei

**Cerințe funcționale**

Pentru ca aplicația cerută să funcționeze și să rezolve problema dorită, aceasta trebuie să îndeplinească următoarele cetințe de bază:

**Extragerea unui polinom dintr-un șir de caractere/introducerea polinoamelor .** Baremul menționează găsirea coeficienților prin Pattern Matching dintr-un text, implicând această funcționalitate pentru utilizator ca principala modalitate de introducere a polinoamelor

**Efectuarea operațiilor pe polinoamele introduse**. Bineînțeles, scopul unui calculator este de a rezolva operații. În cazul acesta, alături de numărul operanzilor ele sunt : adunare, scădere, înmulțire, împărțire, derivare și integrare. De asemenea, utilizatorului trebuie să i se permită să navigheze printre acestea

**Schimbarea dinamică a tipului coeficienților.** În ciuda faptului că enunțul problemei impune coeficienți întregi, în urma operațiilor de împărțire și integrare aceștia pot deveni raționali. Din moment ce enunțul trebuie respectat, nu se pot pune direct coeficienții double

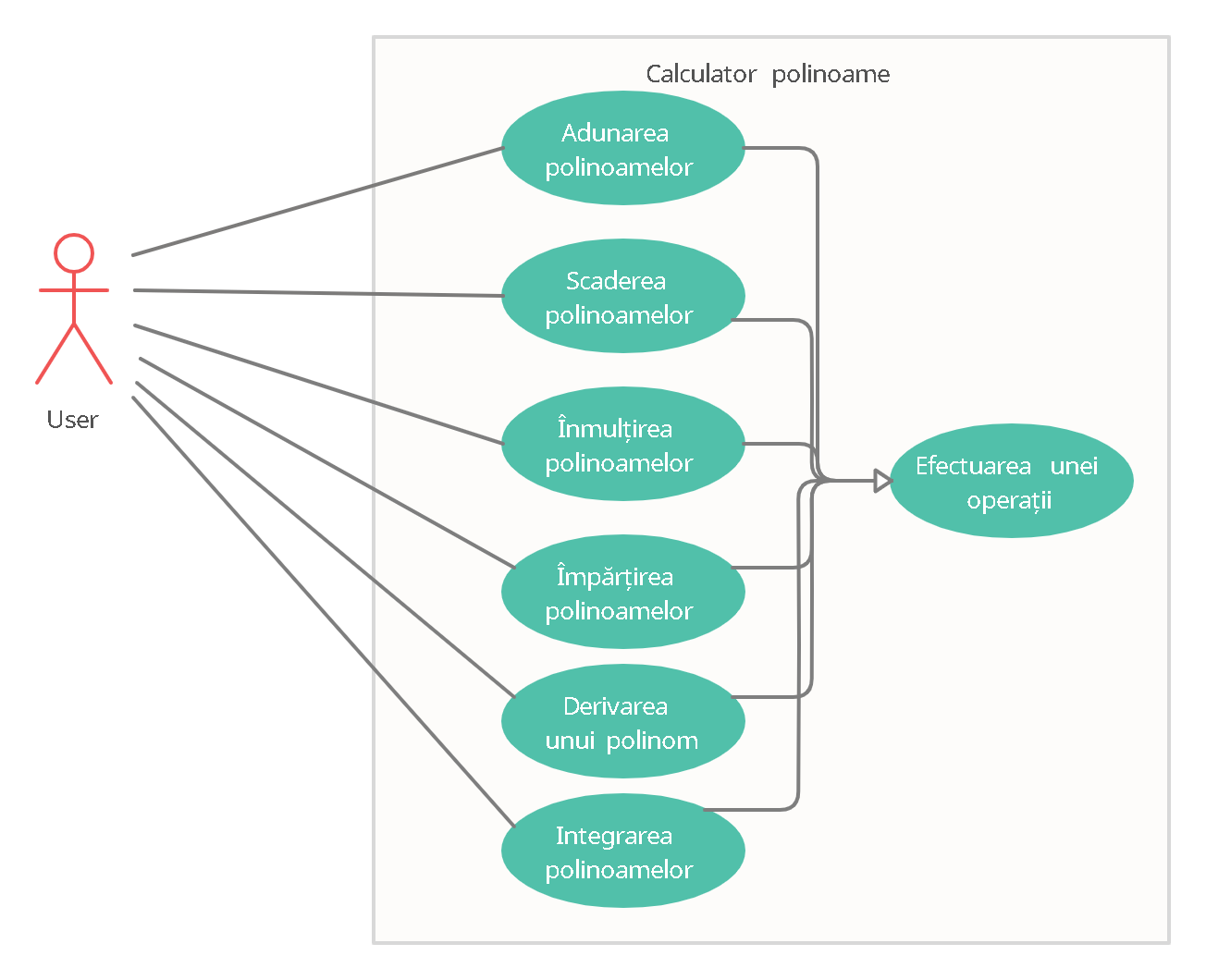
**Cerințe non-funcționale.**

**O interfață grafică ușor de utilizat.** Utilizatorul trebuie să aibă la dispoziție un mod intuitiv de navigare pentru a putea să folosească utilitățile puse la dispoziție fără prea multe tutoriale mai complexe decât aplicația în sine

**Ordonarea termenilor din polinom în ordine descrescătoare după putere și eliminarea termenilor cu coeficientul 0**. În dorința de înfrumusețare a afișării rezultatelor, dar și în simplificarea unelor situații de calcul, se dorește această organizare. Deoarece se impune stocarea polinoamelor în liste în locul Array-urilor, această sortare nu poate fi realizată ușor prin corespondența între niște indici și puteri

**Scenarii de utilizare.**

Avem următoarea diagramă a scenariilor:



Majoritatea cazurilor de utilizare presupun introducerea unor operanzi și efectuarea unei operații. Astfel, în principiu, o utilizare a aplicației are următorul scenariu:

Use case: Orice operație

Actor principal: utilizatorul

Cazul de utilizare principal:

1. Utilizatorul introduce 1 sau 2 polinoame, în funcție de operația pe care o dorește
2. Utilizatorul dă click pe butonul de operație dorit
3. Se afișează rezultatul operației

Scenariu alternativ: Utilizatorul introduce date eronate ( împărțirea cu 0)

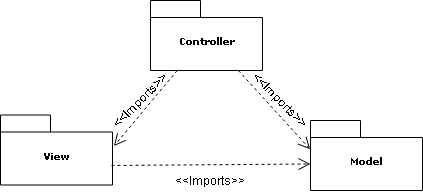
* În acest caz se va afișa un mesaj de eroare în locul rezultatului și se revine la primul pas din scenariul normal

# Proiectare

*Organizarea OOP*

**Pachete**

Pentru o mai bună organizare a codului în scopul mentenanței, acesta trebuie organizat conform unei structuri cunoscute în rândul programatorilor OOP. Din acest motiv, a fost aleasă organizarea Model-View-Controller, structurată în următoarele pachete corespunzătoare fiecărui nivel arhitectural:



În mod normal structura MVC ar avea diagrama de pachete de mai sus, însă în acest caz legătura dintre View și Model nu a fost necesară deoarece Modelul nu reține nimic în mod special din View

Din moment ce proiectul nu este prea complex, nu a fost nevoie de pachete interne

**Clase**

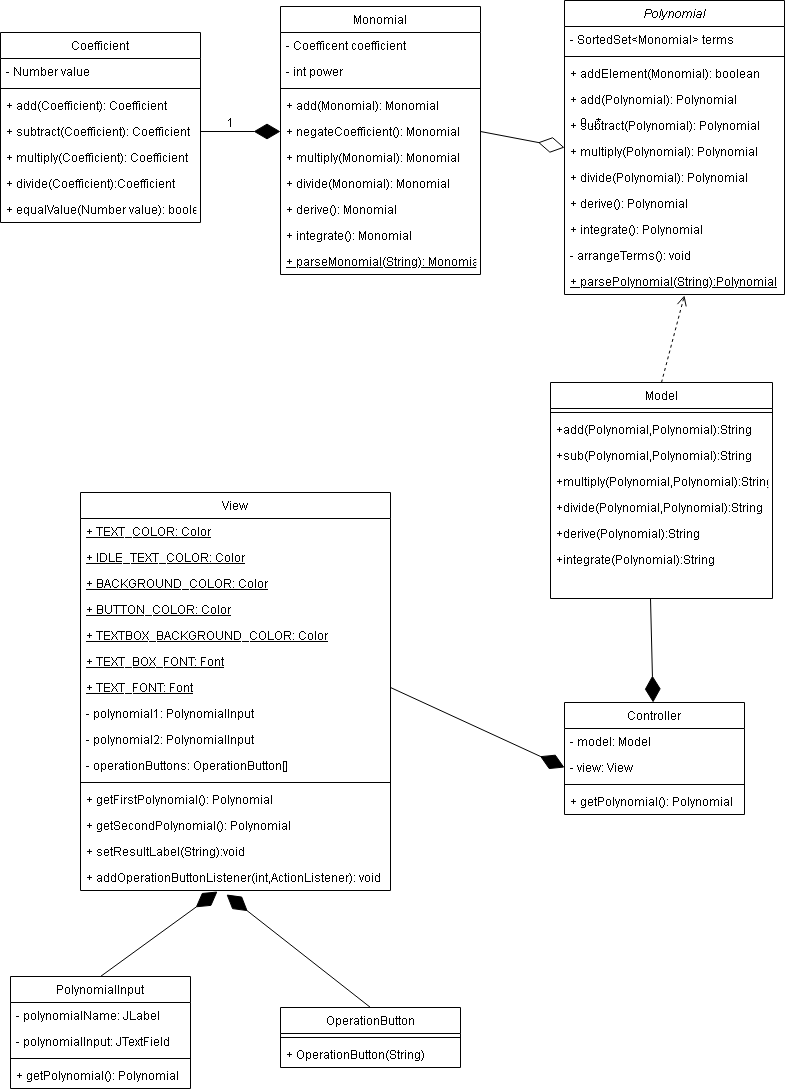
Inițial, singura clasă evidentă ar fi clasa pentru polinom. Considerând însă cerințele funcționale, alături de anumite aspecte teoretice legate de problemă, se pot lua în considerare mult mai multe clase. De exemplu, gândindu-ne la faptul că un polinom este de fapt mai multe monoame, putem desprinde faptul că avem nevoie de o clasă care tratează acest lucru. Având în vedere și nevoia de a schimba între tipurile coeficienților, se pot desprinde două metode de tratare a acestei probleme prin clase suplimentare:

1. Clasă abstractă de monom fără coeficient și clase moștenitoare pentru monoame cu coeficienți de tip int și double. Este o metodă recomandată, însă ar trebui implementate prea multe metode repetitive (de exemplu, scăderea între monoame de care e nevoie în algoritmul de împărțire). De asemenea, ar trebui să se facă foarte mult Downcasting în implementarea operațiilor pe polinoame
2. O singură clasă de monom care conține o clasă coeficient pentru operații pe tipuri numerice generale. Dezavantajul acestei metode e că nici aici nu se scapă de Downcasting, însă nu trebuie să se implementeze atât de multe metode ca în abordarea anterioară. De asemenea, e mult mai ușor să se efectueze operații între monoame cu coeficienți de tipuri diferite și se poate folosi chiar și ulterior în alte probleme în care se cere schimbarea unui tip numeric

În ceea ce privește organizarea metodelor, pentru majoritatea claselor din model, a fost folosită o structură similară cu anumite subclase numerice precum BigInteger sau Decimal în încercarea de a o face mai ușor de folosit. Această structurare include următoarele aspecte:

* Operațiile sunt realizate ca metode în clasa obiectului pe care se efectuează, realizându-se între this și un alt operand dacă e cazul, și returnându-se într-o variabilă de rezultat
* În cazul împărțirii cu rest, se returnează un Array de 2 elemente în care se află câtul și restul pe pozițiile 0 și 1 respectiv
* Există metode de parsare statice în interiorul clasei de parsat

Aceste lucruri, alături de variabile instanță și metode, vor fi observate mai detaliat în [captiolul de implementare](#_Implementare). Până atunci, se poate realiza vederea de ansamblu a claselor prin următoarea diagramă UML:



Interfețe

Nu a fost nevoie de definirea unor interfețe speciale pentru rezolvarea problemei. S-au folosit doar anumite interfețe predefinite din Java precum Comparable (pentru a putea aplica sort pe lista de monoame), Clonable (pentru unele situații în care e nevoie să se copieze polinoame) și, bineînțeles, ActionListener (pentru tratarea evenimentelor din interfața grafică)

*Structuri de date folosite*

Nu prea a fost nevoie de utilizare a unor structuri de date prea complexe în afară de List pentru polinom. Ca tip concret de listă am folosit ArrayList deoarece e rapid pe operațiile pe listă frecvent efectuate în cazul de față

*Algoritmi pentru operațiile pe polinoame*

Rezolvând problema operațiilor pe polinoame, poți realiza că unele lucruri sunt fie mai complexe, fie mai ușoare decât par.

Înainte de a începe prezentarea operațiilor, este nevoie să se prezinte un algoritm foarte important care le simplifică foarte mult. Aceasta este reprezentată de **organizarea termenilor din polinom.** După cum am precizat și în cerințele funcționale, nu putem realiza o sortare și eliminare/ignorare a termenilor nuli (termenii cu coeficient zero) automată ca la un Array. Deci, pentru a ajunge la o formă mai “naturală” a unui polinom, va fi nevoie de execuția următorilor pași:

1. Sortarea termenilor polinomului descrescător după putere
2. Parcurgerea listei sortate și adăugarea termenilor într-o nouă listă după următoarele cazuri

* **Lista e vidă** : se adaugă elementul curent din lista sortată fără alte condiții
* **Puterea termenului curent e egală cu puterea ultimului element adăugat în lista auxiliară**: la valoarea coeficientului ultimului termen se adaugă DOAR valoarea coeficientului elementului curent
* **Puterea termenului curent e diferită de puterea ultimului termen adăugat**: în acest caz trebuie să se adauge termenul curent. Înainte de asta însă, trebuie să se verifice dacă ultimul termen a rămas nul în urma adunărilor și scăderilor dintre coeficienți pentru a fi eliminat

Astfel, în lista auxiliară se va obține polinomul ordonat descrescător cu termenii de aceași putere restrânși și fără termeni nuli. Utilitatea acestei forme va fi prezentată corespunzător pentru fiecare operație care o folosește

**Adăugarea/scăderea**: Chiar dacă ar părea că pentru aceste operații simple ar fi nevoie de algoritmi complicați (precum ceva similar MergeSort-ului), totul se poate reduce la doar a adăuga termenii unui polinom în lista de termeni a celuilalt (înmulțiți cu -1 eventual penru scădere) și aplicarea metodei de aranjare a termenilor

**Înmulțirea**: la fel ca algoritmul clasic de liceu de înmulțire pe foaie, fiecare termen dintr-un polinom se înmulțește pe rând cu fiecare termen din celălalt și eventual se restrâng termenii

**Împărțirea**: din nou, un algoritm care se aplică și pe foaie. Cât timp gradul deîmpărțitului e mai mare sau egal decât gradul împărțitorului, se aplică următorii pași (!presupunând că ambele polinoame au fost aduse în forma organizată przentată anterior):

1. Se împarte primul termen al deîmpărțitului la primul termen al împărțitorului
2. Rezultatul calculului anterior se înmulțește cu împărțitorul
3. Din împărțitor se scade rezultatul înmulțirii

**Derivarea/Integrarea**: doar se derivează/integrează pe rând fiecare termen al polinomului

*Organizarea interfeței grafice*

Un lucru care a durat considerabil în proiectare a fost și organizarea interfeței grafice. În general, este foarte important ca o interfață grafică să fie plăcută atât în mod estetic, cât și în mod utilitar. Acest lucru înseamnă ca nu numai aplicația trebuie să arate bine, însă și anumite componente trebuie organizate și scoare în evidență în scopul facilitării folosirii

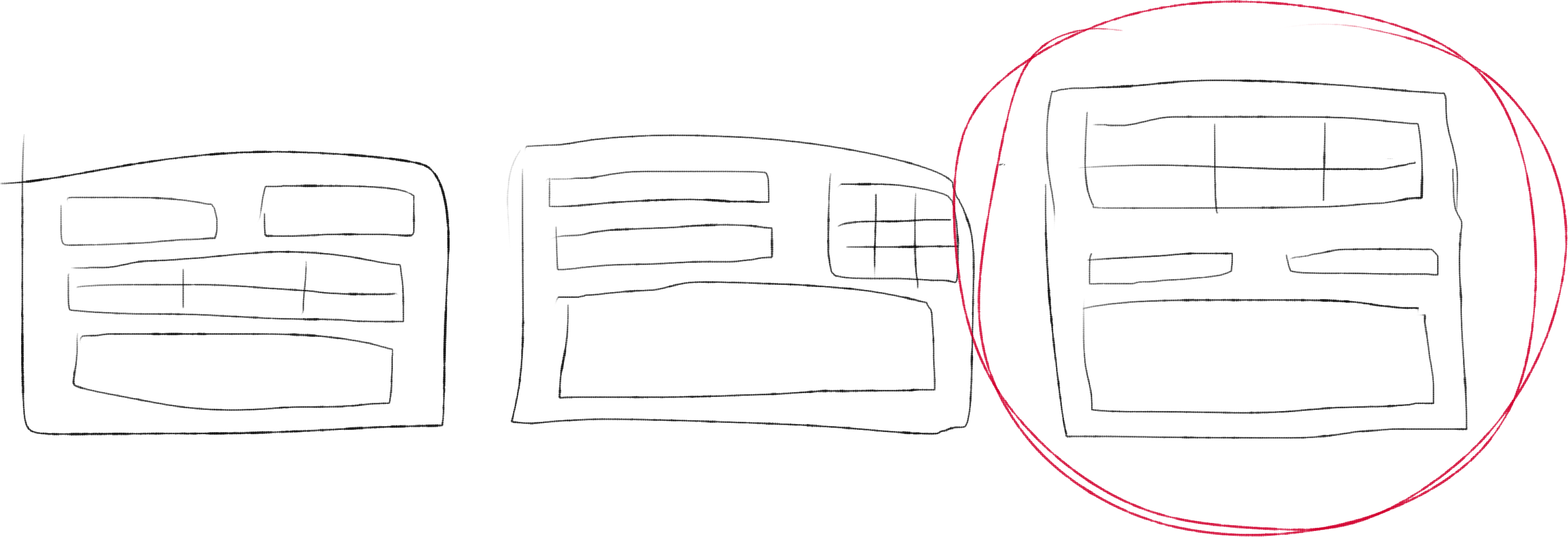
Înainte de organizarea elementelor din interfața grafică (butoane, casete de text etc.), acestea trebuie în primul rând identificate. Acest pas ajută la stabilirea variabilelor instanță din clasa de View destinate componentelor grafice, alături de tipurile lor. În funcție de cât de mult trebuie să atragă atenția, am obținut următoarele componente în următoarea ordine:

Zona de rezultat – Zonă de text care trebuie să fie foarte mare pentru că asta îl interesează în primul rând pe utilizator. De asemenea, aceasta poate conține polinoame foarte lungi ai căror termeni trebuie vizualizați simultan

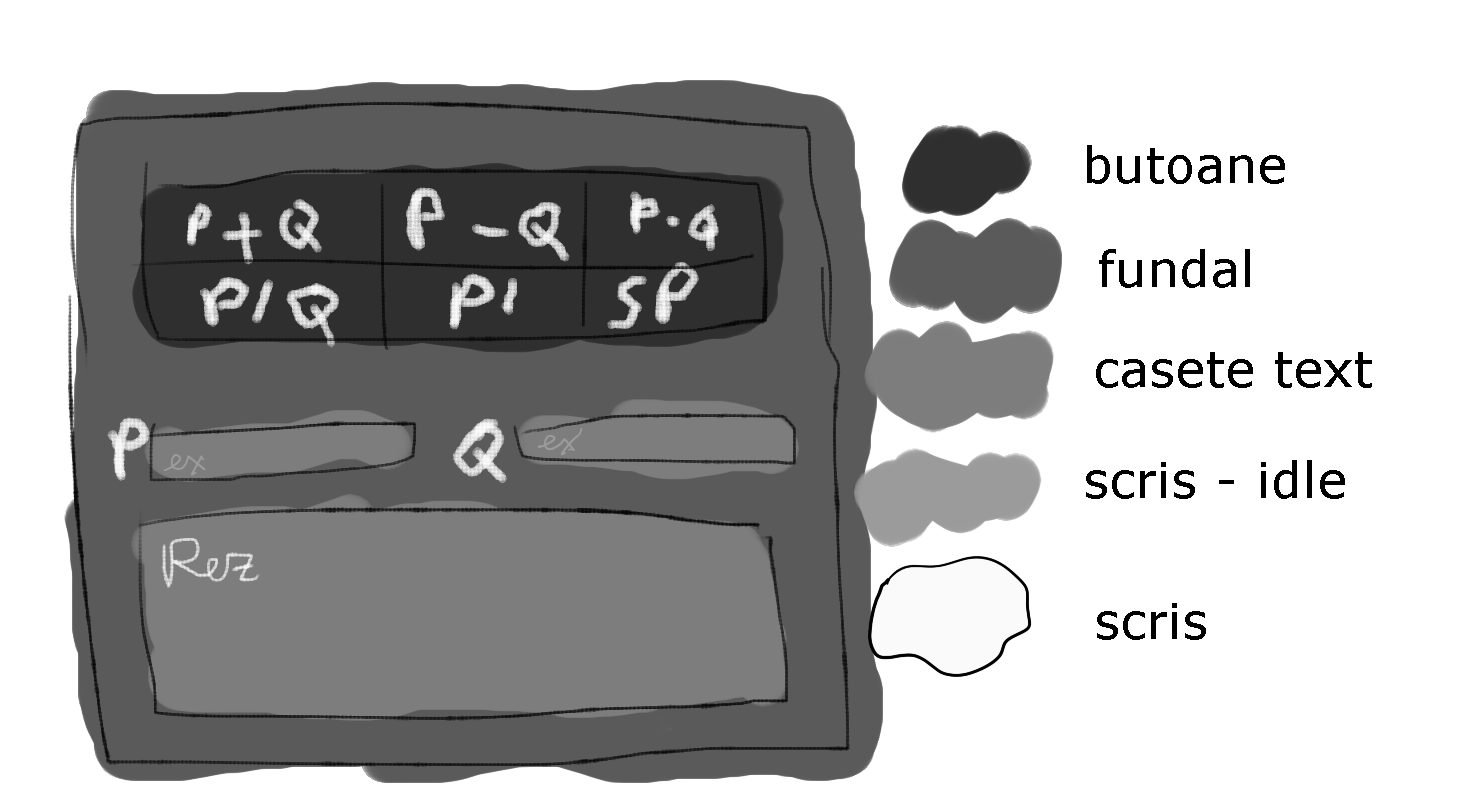
Zonele pentru operanzi – Zone de text care pot fi și pe un singur rând

Butoanele de operații – 6 butoane. Câte unul pentru fiecare operație

Cunoscând aceste elemente, se realizează apoi așezarea în pagină sub formă de miniatură. Chiar dacă acst lucru implică precizie în așezare, este nevoie și de creativitate pentru a găsi un mod estetic. De aceea am schițat cu mâna liberă mai multe lucruri, până când am ajuns la ceva ce mi-a plăcut



După alegerea unei poziționări convenabile pentru stabilirea Layout-urilor, am realizat o mapare alb-negru a culorilor. Aceasta constă în atribuirea unei culori alb-negru fiecărei componente în funcție de contrastul/luminozitatea pe care o vor avea pentru a vedea ce elemente grafice ies în evidență și pentru a vedea de câte culori diferite este nevoie în realizarea aplicației pentru a stabili anumite constante de culoare pentru modificarea ușoară a esteticii



# Implementare

*Pachetul model*

Pachetul pentru clasele ce implementează operațiile interne

* ***Clasa Coefficient***. Aceasta reprezintă modul de tratare al modificării tipului coeficienților din int în double în urma operațiilor de împărțire și integrare

Câmpuri: Number value: valoarea numerică efectivă a coeficientului, care este fie Integer sau Double

Metode: Cele mai importante sunt metodele de operații (add, dif,multiply,divide), care tratează operațiile între numere de tipuri diferite prin realizarea de Downcasting la clasa de Wrapper a valorii. Acestea au fost realizate datorită faptului că operatorii nu se pot aplica direct pe clasa de Number, fiind nevoie de tipul exact al operanzilor.

Pe lângă aceste metode, există metode de comparare (equalsValue și compareTo) pentru eventualele operații de comparare, în special din cauza faptului că în cazul numerelor Double poate exista o eroare în reprezentare.

* ***Clasa Monom***. Aceasta este clasa pentru termenii polinomului, având la rândul ei metode pentru scurtarea codului din clasa pentru polinom. De asemenea, ea implementează Comparable pentru organizarea termenilor polinomului prin metoda Collections.sort()

Câmpuri:Coefficient coefficient: coeficientul monomului

int power: puterea monomului

Metode importante: Pe lângă metodele de operații pe monoame(add, multiply, divide, derive,integrate), alte metode importante sunt:

compareTo :folosită în aranjarea termenilor din polinom descrescător după putere

parseMonomial: metodă statică folosită pentru extragerea unui polinom cu coeficient întreg dintr-un șir sursă. Respectarea unui pattern nu este verificată aici deoarece se realizează deja în funcția de parsare a polinoamelor

* ***Clasa Polinom***. Clasa care tratează partea internă a problemei în sine (rezolvarea operațiilor pe polinoame). În ea se găsesc de asemenea metode ajutătoare pentru diverse lucruri.

Câmpuri: List<Monomial> terms : termenii polinomului sub formă de listă

Metode importante: Pe lângă metodele pentru operațiile cerute în enunț (addPolynomials, subtractPolynomials,multiplyPolynomials,dividePolynomials,derivePolynomials și integratePolynomials), alte metode mai importante sunt:

arrangeTerms: după cum îi spune și numele, această metodă aranjează termenii polinomului, după modul menționat în capitolul anterior. Această metodă este foarte utilizată în metode interne, fiind metodă privată

parsePolynomial: metodă statică pentru extragerea unui polinom cu coeficienți întregi dintr-un text. Doar se caută pattern-ul de coeficienți - (([+-])?([0-9])\*((x)((\^)([0-9])+)?)?) – și se aplică parsarea de Monom pentru fiecare match găsit

* ***Clasa Model***. În mare parte doar formatează rezultatele operațiilor (de exemplu: pune „+C” la rezultatul integrării, pune câtul și restul împărțitii împreună). Nu are variabile instanță, iar toate metodele sale sunt statice și doar aplică operațiile de bază din polinom

*Pachetul Controller*

Doar realizează legătura dintre interfața grafică și operațiile interne prin clasa de Controller și clase interne acesteia care implementează ActionListener (AddListener)

* ***Clasa Controller***

Aici doar se face legătura dintre interfața grafică și operațiile pe polinoame. Acest lucru este realizat cu clase interne cu nume sugestive care implementează ActionListener (și, bineînțeles, metode specifice pentru actionPerformed):

* AddListener
* DifListener
* MultiplyListener
* DivListener
* DeriveListener
* IntegrateListener
* De asemenea, aici se găsesc variabile instanță pentru Modelul și interfața grafică între care se face legătura

*Pachetul View*

Tratează implementarea interfeței grafice. Din moment ce aceasta nu este prea complexă, nu prea există multe clase pe lângă clasa cu interfața finală

* ***Clasa PolynomialInput.*** Clasa pentru câmpurile de introducere a polinoamelor pentru operații

Câmpuri: JLabel polynomialName: Reține numele polinomului (de exemplu P(x))

JTextField polynomialInput: Pentru scrierea propriu-zisă a polinomului. Are implicit un text cu exemplu de input pentru polinom și un FocusListener anonim pentru eliminarea exemplului la scriere

String dummyText: O constantă privată pentru un exemplu de introducere al polinoamelor

Metode: getPolynomial preia textul din polynomialInput și îl returnează în formă de polinom prin parsare

* ***Clasa OperationButton***. Clasa pentru butoanele de operații. Doar se formatează butoanele în constructor
* ***Clasa View***. Clasa pentru fereastra finală a aplicației

Câmpurile din această clasă pot fi clasificate în două tipuri:

Constante de culoare/font. Alegerea unei tematici nu este chiar atât de ușoară pe cât pare. Uneori trebuie experimentat foarte mult până la atingerea unui rezultat dorit. De aceea, pentru a controla mai ușor culorile, doar am ales câteva culori de bază pe care le-am mapat la câte o componentă, după care am modificat direct valoarea din constantă când a venit timpul să aleg aspectul estetic

TEXT\_COLOR: Culoarea pentru scrisul din butoane,caseta de text pentru rezultat si field-urile de text de polinoame

IDLE\_TEXT\_COLOR: Culoarea pentru exemplul care apare înainte de scrierea polinomului dorit

BACKGROUND\_COLOR: Culoarea pentru fundalul în sine al aplicației

BUTTON\_COLOR: culoarea de fundal

TEXTBOX\_BACKGROUND\_COLOR: Culoarea de fundal a casetelor de text pentru rezultat și operanzi

TEXTBOX\_FONT: pentru fontul din polynomialInput și panoul de text de rezultat

TEXT\_FONT: pentru fontul din butoane

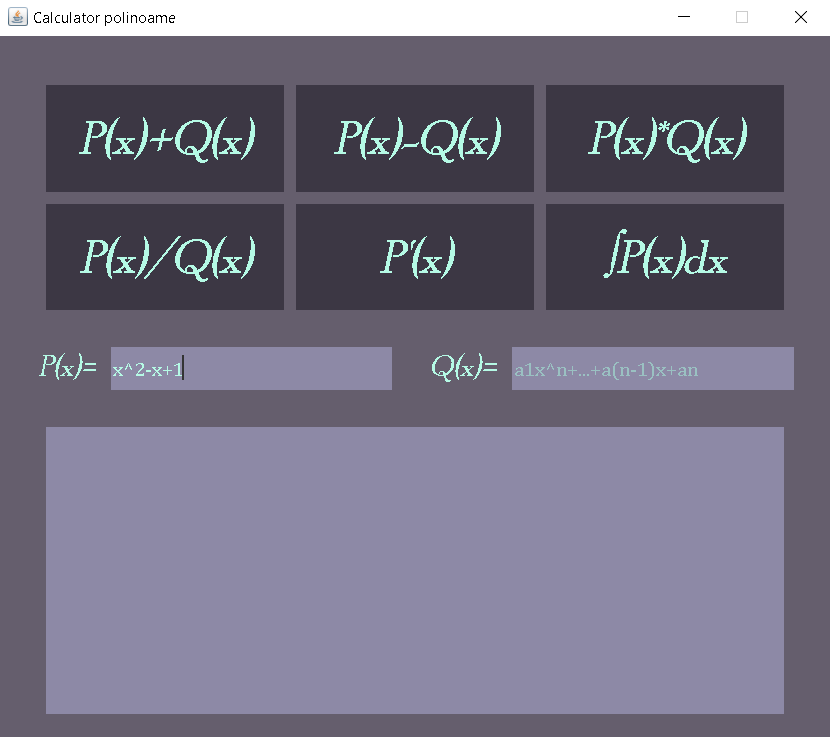
Componentele ferestrei:

JTextPane resultLabel: Câmpul de text needitabil în care se vizualizează rezultatul

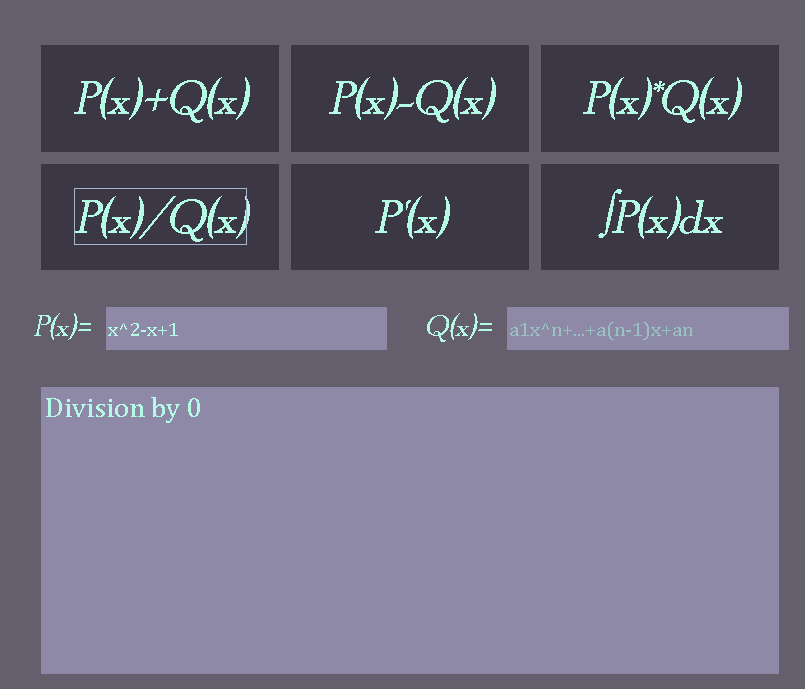
PolynomialInput polynomial1Field,polynomial2Field: Câmpurile de text pentru introducerea polinoamelor

OperationButton[] operationButtons: Un Array pentru toate butoanele de operație

*Intefrața grafică*

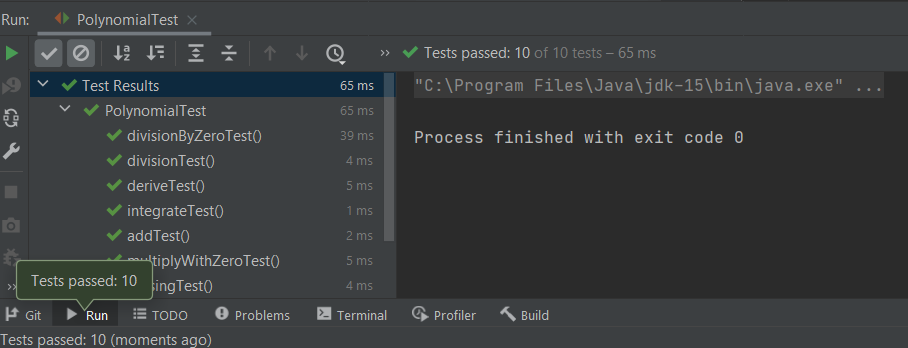


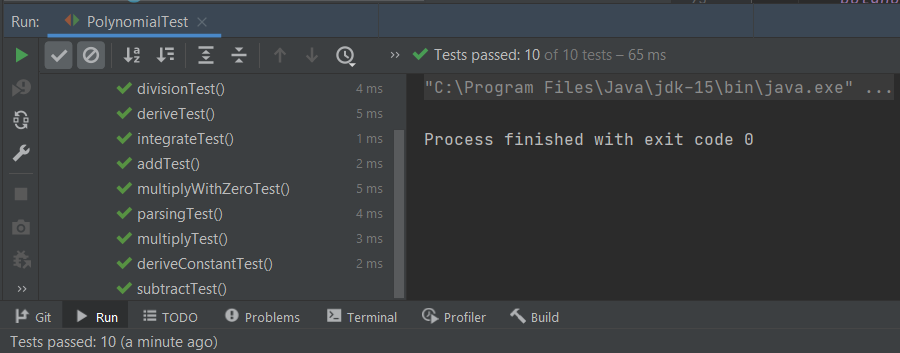
Interfața este implementată în Java Swing urmând pașii menționați în capitolul anterior. Aceasta are un aspect simplist și intuitiv: Scrii polinoamele dorite în câmpurile specifice, după care apeși pe butonul corespunzător operației dorite, lucru care va determina automat afișarea rezultatului. Dacă într-un câmp nu s-a scris nimic, atunci este considerat ca fiind 0



Testare și rezultate

Pentru testarea unitară a operațiilor pe polinom s-a folosit Framework-ul Junit. În principiu, a fost realizat câte un test pentru fiecare operație, alături de puține teste pe cazuri speciale (împărțire și înmulțire cu 0, derivare de constantă) și un test de parsare a polinomului





# Concluzii

Acest proiect m-a ajutat atât să îmi mențin cunoștințele de Programare Orientată pe Obiect, Java și Interface Design deja existente, cât și să îmi îmbunătățesc abilitățile de abstractizare,organizare și încapsulare a datelor. A fost un proiect nu prea complicat, însă nici realizabil într-o singură noapte, bun pentru reamintirea cunoștințelor necesare, introducerea în această materie, dar și învățarea de utilizare a unor noi lucruri în Java chiar foarte utile pentru proiecte ulterioare (partea de Pattern si Match). Pe lângă informațiile noi învățate din prezentarea de îndrumare, am fost de asemenea motivată să caut informații suplimentare despre clase și pachete din Java foarte utile(GroupLayout, DecimalFormat).

# Îmbunătățiri ulterioare

**Reducerea complexității operației de înmulțire**. Din câte am înțeles, ar fi mult mai eficient ca înmulțirea să se efectueze printr-o formulă ce include transformata Fourier. Din păcate, asta include și îmbunătățirea cunoștințelor de matematică ale proiectantului

**Mesaje de eroare pentru parsare incorectă**. În moentul actual, parsarea de polinoame doar identifică un pattern într-un șir de caractere, ignorând eventuale caractere în plus introduse în șirul sursă. Chestia aceasta nu este un lucru neapărat oribil, și nici nu a fost o cerință obligatorie, însă s-ar putea detecta ușor la început dacă un șir respectă în primul rând pattern-ul de polinom

**Posibilitatea operării pe rezultate intermediare (buton de „ans” pentru preluarea rezultatului din caseta de rezultat).** Inițial, chiar aveam de gând să implementez această funcționalitate (de aceea am și optat pentru o clasă de coeficient în locul a două clase de monoame diferite), însă nu am vrut să gândesc unde să pun un buton pentru așa ceva

**Buton de swap între operanzi.** Nu ar fi greu de implementat. Doar ar trebui reținuți operanzii în clasa de model

# Bibliografie și resurse

<https://docs.oracle.com/javase/tutorial/uiswing/layout/group.html>

<https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html>

Pentru diagrame:

UML: <https://www.draw.io/>

Use-Case: <https://creately.com/>