**Calculator de polinoame. Documentație**

Grigorescu M. Alexandru

Departamentul de calculatoare, Universitatea Tehnică, 30223

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**DEPARTAMENTUL CALCULATOARE**



Tehnici de programare

Prof. dr. ing. Tudor Cioara

21 Martie, 2024

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc95297886)

[3. Proiectare 3](#_Toc95297887)

[4. Implementare 3](#_Toc95297888)

[5. Rezultate 3](#_Toc95297889)

[6. Concluzii 3](#_Toc95297890)

[7. Bibliografie 3](#_Toc95297891)

# Obiectivul temei

Este dorită elaborarea unei aplicații dedicate manipulării polinoamelor, o componentă esențială în studiul matematicii și în numeroase domenii tehnice și științifice. Pentru realizarea proiectului trebuie respectate următoarele etape esențiale pentru a asigura dezvoltarea eficientă și eficace a aplicației.

### Determinarea unor cazuri de utilizare

Cazurile de utilizare (sau usecase-urile) sunt scenarii specifice în care utilizatorii vor interacționa cu sistemul nostru și vor utiliza funcționalitățile oferite de acesta. Prin definirea și înțelegerea acestor usecase-uri, putem ghida procesul de dezvoltare și ne asigurăm că aplicația noastră va satisface nevoile utilizatorilor într-un mod complet și eficient. Informații suplimentare și detaliate sunt furnizate în cadrul Capitolului 2, unde subiectul este explorat în profunzime.

### Proiectarea algoritmilor de prelucrare a polinoamelor

Procesul de proiectare a algoritmilor pentru manipularea polinoamelor constituie prima etapă în dezvoltarea aplicației propuse. Această etapă implică identificarea și definirea operațiilor de bază, adunarea, scăderea, înmulțirea, împărțirea, derivarea și integrarea polinoamelor. În plus, este necesară analiza eficienței și optimizarea algoritmilor pentru a asigura o performanță optimă a aplicației în condiții diverse de utilizare. Informații suplimentare și detaliate sunt furnizate în cadrul Capitolului 2, unde subiectul este explorat în profunzime.

### Modelarea tipurilor de date utilizate

Alegerea unor structuri de date adecvate este crucială pentru eficiența algoritmilor implementați. De asemenea, este important să se țină cont de aspectele legate de gestionarea memoriei și de eficiența în timpul de execuție. Informații suplimentare și detaliate sunt furnizate în cadrul Capitolului 2, unde subiectul este explorat în profunzime.

### Crearea unei arhitecturi flexibile

O arhitectură bine definită și flexibilă reprezintă o condiție prealabilă pentru o dezvoltare rapidă și eficientă a aplicației. Prin adoptarea unor principii de proiectare software, cum ar fi modularitatea, coeziunea și cuplajul redus, se asigură o structură a codului care poate fi ușor extinsă și modificată în viitor, în funcție de cerințele și feedback-ul utilizatorilor. Informații suplimentare și detaliate sunt furnizate în cadrul Capitolului 3, unde subiectul este explorat în profunzime.

### Implementarea unei interfețe grafice prietenoase cu utilizatorul

Interfața grafică reprezintă componenta prin care utilizatorii interacționează cu aplicația și, în consecință, joacă un rol crucial în experiența lor de utilizare. Crearea unei interfețe grafice intuitive și prietenoase, care să faciliteze utilizatorilor accesul și utilizarea funcționalităților oferite de aplicație, necesită o atenție deosebită acordată designului ergonomic, navigației intuitive și feedback-ului vizual. Informații suplimentare și detaliate sunt furnizate în cadrul Capitolului 3, unde subiectul este explorat în profunzime.

# Analiza problemei, modelare, scenarii, cazuri de utilizare

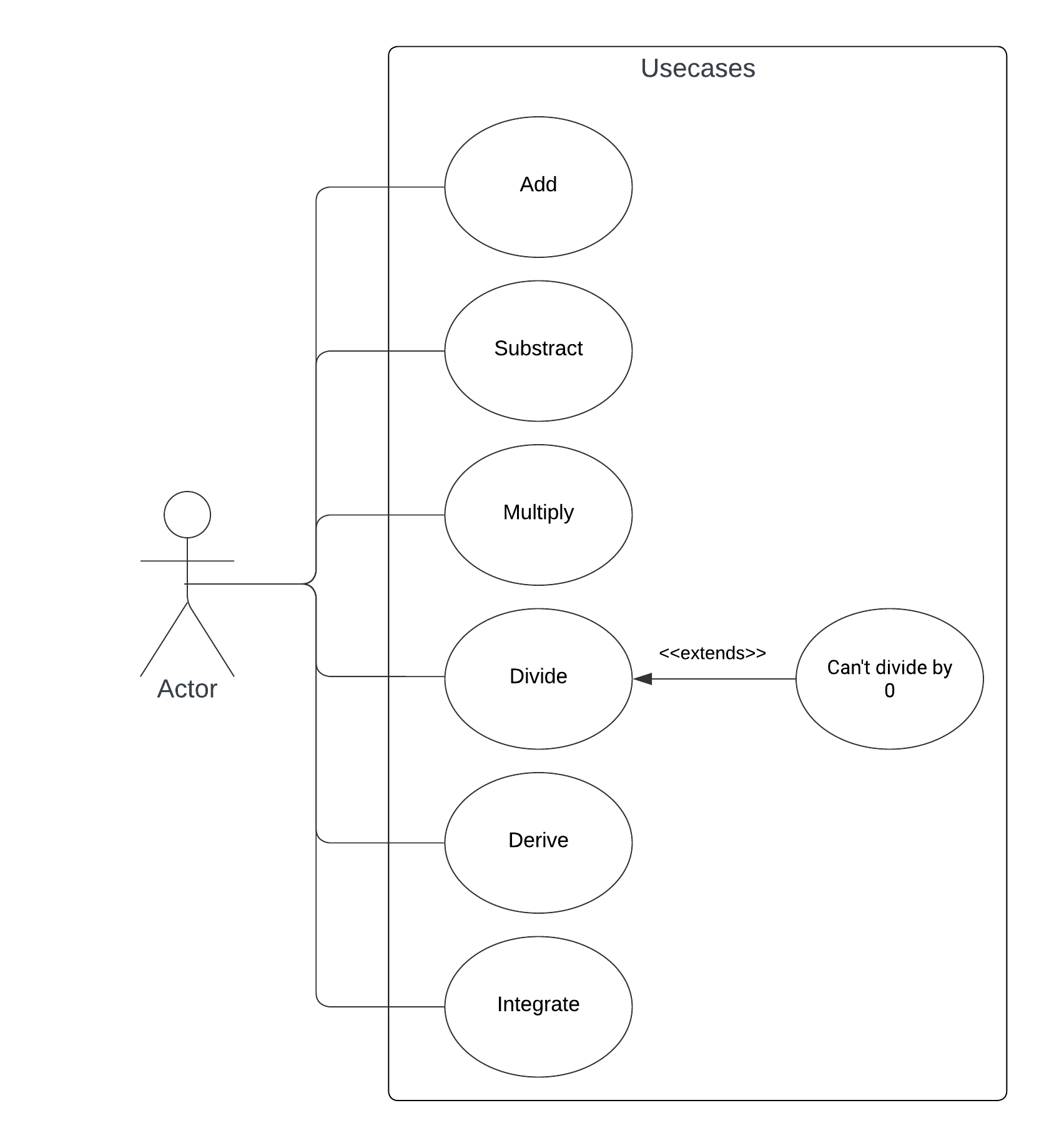
### Cerințe Funcționale

Aplicația *calculatordepolinoame.ro* trebuie să ofere următoarele servicii:

* Trebuie să permită utilizatorului să introducă unul sau două polinoame
* Trebuie să permită utilizatorului să aleagă tipul de operație matematică
* Trebuie să adauge două polinoame
* Trebuie să scadă două polinoame
* Trebuie să înmulțească două polinoame
* Trebuie să împartă două polinoame
* Trebuie să deriveze un polinom
* Trebuie să integreze un polinom
* Trebuie să trateze cazurile de excepții

### Cerințe non funcționale

Aplicația *calculatordepolinoame.ro* trebuie să aibă următorul comportament:

* Trebuie să informeze utilizatorul în cazul erorilor de sistem
* Trebuie să aibă o interfață prietenoasă cu utilizatorul
* Trebuie să poată returnă rezultatul oricărui tip de ecuație în sub o secundă
* Trebuie să funcționeze pe orice tip de calculator/laptop
* Trebuie să fie construit modular, astfel încât adăugarea noilor funcționalități să fie rapidă

### Cazuri de utilizare

Diagrama alăturată prezintă toate cazurile de utilizare tratate de *calculatoruldepolinoame.ro.* În continuare se poate observa cum fiecare request al utilizatorului este procesat de sistemul nostru. În dorința de îmbunătății cât mai mult experiența utilizatorului, sistemul nostru tratează orice input invalid drept polinomul “0”.

## 

### Proiectarea algoritmilor de prelucrare a polinoamelor

Algoritmii de prelucrare a polinoamelor sunt realizați după cum urmează. Pentru simplitate în discurs ne vom referi la primul polinom drept P și la al doilea drept Q. Monomul este cea mai mică parte a unui polinom, el este compus din exponent și coeficient.

#### Adunarea, scădere

Pentru a aduna două polinoame am însumat coeficienții monoamelor de același grad. Analog am scăzut din coeficientul monomului lui P coeficientul monomului lui Q.

#### Înmulțirea

Pentru realizarea acestei operații trebuie să înmulțim pe rând fiecare element din P cu fiecare element din Q.

#### Împărțirea

Operația de împărțire poate să returneze atât câtul împărțirii, cât și restul acesteia. Arhitectura sistemul (vezi cap. 3) filtrează cererile de calcul pentru împărțiri astfel încât sunt evitate cazurile în care împărțitorul este 0 sau câtul urmează sa fie 0 (rangul lui P este mai mic decât cel al lui Q). Pentru a calcula restul împărțim pe primul monom al lui P cu primul monom al lui Q. Acest rezultat (R) este înmulțit cu Q și scăzut din P. P ia valoarea ultimei operații, iar procesul se repetă până când rangul lui P este mai mic decât al lui Q. În acest moment, valoarea restului este salvată în P. Câtul urmează aceeași regulă, însă de data asta valorile lui R sunt adăugate într-un nou polinom, polinomul cât.

#### Derivarea, integrarea

Pentru efectuarea acestei operațiuni, este parcurs fiecare monom în parte și sunt aplicate regulile de derivare, respectiv integrare.

### Modelarea structurilor de date

Odată ce am definit toate operațiile în secțiunea anterioară putem înțelege cum ar trebui să se comporte un polinom. Știm că este format dintr-o listă de monoame și are un rang. Totodată din usecase-uri putem trage concluzia că pentru a crea un polinom avem nevoie de un șir de caractere care îl definește matematic. Cât despre comportament știm următoarele: trebuie să își poate seta și obține monoamele și rangul.

Din diagrama UML se observă în plus două metode care în esență fac același lucru, însă au implementări diferite: *updateRank*(calculează singur rangul polinomului O(numărul de monoame)) și *setRank*(setează rangul la o valoarea specificată O(1)).

Argumentul împotriva instituirii unei clase specializate *Monomial* în contextul matematicii se întemeiază pe stabilitatea și durabilitatea definițiilor matematice. În matematică, definițiile sunt adesea considerate entități stabilite și puțin susceptibile la schimbări. Prin urmare, este puțin probabil ca o clasă dedicată *Monomial* să fie modificată în timp.

În plus, alegerea de a nu implementa o clasă specializată pentru *Monomial* este justificată și de simplitatea soluției alternative propuse. Această alternativă constă în utilizarea unui simplu *HashMap*, care oferă o implementare eficientă și ușor de înțeles pentru gestionarea monoamelor.

Prin urmare, prin evitarea unei clase specializate în favoarea unei soluții mai generice și simplificate, se obține o structură mai ușor de întreținut și de înțeles în contextul matematic.

# A diagram of a computer system Description automatically generatedProiectare

Întrucât aplicația dispune de o interfață grafică am ales o arhitectură de tipul MVC. Modul în care datele sunt transmise de la utilizator la aplicație și înapoi la utilizator este descris în diagramă de mai jos (figura 1). În următoarele secțiuni voi detalia modul de funcționare al sistemului

### Routing

A diagram with text and a black arrow

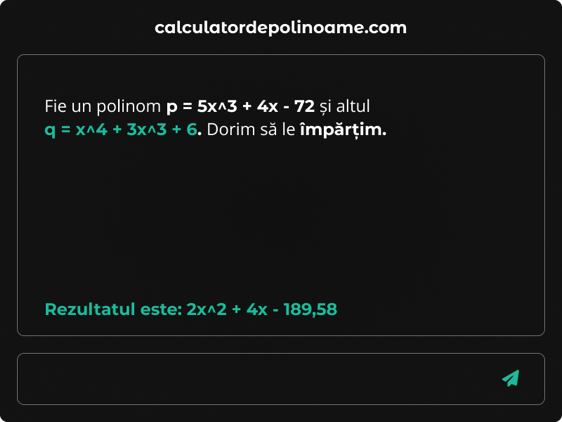
Description automatically generated with medium confidencePentru a asigura o dezvoltare rapidă pe viitor, am creat un sistem de rutare care permite adăugarea rapidă și ușoară a rutelor noi. În cadrul aplicației, există o singură instanță a Router-ului care poate fi accesată din orice loc. Procesul începe odată ce cererea este trimisă către Router, care o direcționează către ruta corespunzătoare. Apoi, cererea este trecută către o clasă numită ControllerRunner, care, la rândul său, apelează metoda action din controler, având cererea ca parametru. Detaliile acestei implementări sunt prezentate în figura 3. Prin acest sistem, se îmbunătățește flexibilitatea și eficiența în gestionarea rutelor și a controlerelor, permițând o dezvoltare agilă și o adaptare rapidă la cerințele viitoare.

### Interfața

Interfața a fost proiectată folosind Figma, o unealtă ce permite crearea rapidă și ușoară a design-urilor. Aplicația poate fi în una dintre cele trei stări: inițială, în curs de completare, sau în stare de succes sau eroare. Abordarea în crearea design-ului a fost minimalistă, evitând supraîncărcarea utilizatorului cu toate opțiunile disponibile încă de la început. În schimb, sunt prezentate doar opțiunile relevante pentru stadiul curent al aplicației. Pentru a îmbunătăți experiența utilizatorului, am inclus și o schemă de culori pentru text, care reprezintă polinomul în curs de scriere. La finalizarea procesului, utilizatorul își selectează operația dorită prin intermediul unui meniu drop-down. Orice rezultate sau erori generate sunt afișate la finalul secțiunii de text. Aceste elemente A screenshot of a black background

Description automatically generatedA black screen with white text

Description automatically generatedcontribuie la o experiență mai plăcută și mai intuitivă pentru utilizator.

A screenshot of a black and white screen

Description automatically generated

### Integrarea interfeței

Pentru crearea interfeței, s-a utilizat JavaFX, menținând consistența în culori și formatare cu ajutorul unor clase ajutătoare: *Colors* (care salvează culorile personalizate folosite în proiect, cum ar fi roșu și verde) și *ComboBoxOptions* (deoarece un *ComboBox* nativ nu poate avea valori separate pentru textul afișat pe ecran și valoarea folosită în cerere, această clasă permite acest lucru).

Conform metodologiei JavaFX, view-ul este construit din două părți: fișierul .fxml și o clasă Controller care se ocupă de interacțiunea cu aplicația. În acest controller, sunt definite variabile pentru câmpurile de text pentru P și Q (polinoame), mesajul de eroare, ComboBox și input-ul utilizatorului. În ComboBox sunt enumerate operațiile posibile pe care le poate realiza utilizatorul. Atunci când se apasă pe câmpul unui polinom, acesta devine selectat, inputul primește valoarea acestuia și culoarea se schimbă în verde. Atunci când butonul de trimitere este apăsat, este creată o cerere și este trimisă către router (vezi secțiunea de Routing).

A screenshot of a computer

Description automatically generatedA white text box with black text

Description automatically generatedAceastă interfață este integrată în restul aplicației prin intermediul unui sistem de view-uri. Acest sistem are la bază o clasă *Window* de tip singleton care oferă un API pentru controllere, prin intermediul căruia pot fi setate view-urile după numele acestora. Pentru a putea transmite date de la controller la view, a fost creată o clasă Session care se comportă similar cu cea implementată de browser.

# Rezultate

*Se vor prezenta scenariile pentru testare. In cazul in care in cerinta temei se specifica sa se faca testare unitara cu utilitarul JUnit, se vor integra in acesta sectiune rezultate testari cu JUnit.*

# Concluzii

*Se vor prezenta concluziile, ce s-a invatat din tema, posibile de dezvoltari ulterioare.*

# Bibliografie

*Se vor adauga referintele care au fost consultate de student pe parcursul implementarii temei .*

*Exemplu:*

1. *Bruce Eckel, Thinking in Java (4th Edition), Publisher: Prentice Hall PTRUpper Saddle River, NJUnited States, ISBN:978-0-13-187248-6 Published:01 December 2005.*
2. *What are Java classes? -* [*www.tutorialspoint.com*](http://www.tutorialspoint.com)