UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

CALCULATOR DE POLINOAME

Facultatea de Automatică și Calculatoare
Departamentul Calculatoare
Curs: Tehnici de Programare Fundamentale
An universitar 2020-2021

Student:
Bîrluțiu Claudiu-Andrei
CTI-ro, An 2, Grupa 30226

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Cuprins

1. Problema și soluția dată

Operațiile matematice ce se execută pe polinoame nu sunt grele, dar consumă mult timp. De asemenea, necesită o atenție deosebită pentru a constrânge monoamele de același grad în cadrul polinomului rezultat, astfel încât să nu avem monoame de același grad duplicate în forma finală a rezultatului. Operația de împărțire necesită mai mulți pași de executat, iar făcând totul pe foaie este destul de greu de urmărit și aproape inevitabil să nu greșești un coeficient sau un grad al unui monom.

Soluția adusă acestei probleme este un calculator de polinoame, care permite prin intermediul unei interfețe grafice să se introducă polinoame și să se afișeze rezultatul operației selectate.

2. Obiectivul temei

Obiectivul principal se referă la proiectarea și implementarea unui calculator ce realizează operații pe polinoame, utilizându-se o interfață grafică prin care un utilizator poate să introducă polinoame și să selecteze operația matematică de executat. De asemenea, prin intermediul interfeței grafice, utilizatorul poate să vizualizeze rezultatul operației scris sub o formă ușor de înțeles și interpretat.

Obiectivele secundare, ce reprezintă pașii de urmat în îndeplinirea obiectivului principal, se regăsesc în următorul tabel. Acestea vor fi abordate în amănunt în următoarele capitole ale acestui document.

Obiectiv	Scurtă descriere	Capitol
Analiza problemei și identificarea cerințelor	-analiza se referă la descompunerea problemei în componente mai mici prin procesul de abstractizare. -se poate vorbi și de o inginerie a cerințelor care stabilește serviciile cerute sistemului precum și constrângerile.	3. Analiza problemei, modelare, scenarii, cazuri de utilizare
Proiectarea calculatorului de polinoame	-obținerea unui sistem bun prin utilizarea unor resurse existente și noi resurse. -reprezintă o activitate de tipul divide-and-conquer. -presupune folosirea paradigmei OOP, realizarea de diagrame UML, proiectarea claselor etc.	4. Proiectare
Implementarea calculatorului de polinoame	-presupune dezvoltarea și scrierea codului în limbajul de programare Java pentru clasele și metodele acestora.	5. Implementare
Testarea calculatorului	-testare la mai multe nivele. -testarea corectitudinii rezultatelor pentru operațiile pe polinoame și monoame în diferite scenarii - folosirea JUnit-ului	6. Rezultate

Tabel 1-Obiective secundare

3. Analiza problemei, modelare, scenarii, cazuri de utilizare

3.1 Analiza problemei

Este important să înțelegem problema și domeniul ei. Procesul de analiză are drept scop descompunerea problemei mari în componente mici și de înțeles.

Domeniului problemei prezente se concentrează în jurul modelului matematic de polinom. În continuare voi prezenta câteva aspecte teoretice legate de ideea de polinom, adaptate la specificațiile din enunțul problemei (polinoame de o singură variabilă-X).

Un polinom de gradul n în nederminată X se scrie în forma canonică (algebrică) astfel:

$$P(X) = a_0X^n + a_1X^{n-1} + ... + a_{n-1}X + a_n, \quad \text{unde } a_0 \neq 0.$$

,unde numerele a_0, a_1,... a_n sunt coeficienții polinomului *Equation 1 Forma algebrică a unui polinom*

Se numește monom, un polinom care are un singur coeficient nenul. În acest caz, un polinom poate fi văzut ca fiind construit din unul sau mai multe monoame. Scrierea unui polinom sub formă algebrică este unică, abstracție făcând de ordinea de scriere a monoamelor.

Astfel, un monom are forma a_iX^k, unde a_i reprezintă coeficientul, iar k reprezintă gradul monomului.

Un termen constant (de exemplu: P(X)=4) are gradul 0.

Gradul unui polinom se definește ca fiind cel mai mare grad al termenilor săi.

Termenul dominant al unui polinom este termenul/monomul care dă gradul polinomului.

Un polinom este nul, dacă toți coeficienții săi sunt nuli. Cu alte cuvinte, termenul dominant al său este monomul 0.

Proprietăți alea polinoamelor

- Suma/diferența a două polinoame este un polinom
- Produsul a două polinoame este un polinom
- Derivata unui polinom este un polinom
- Primitiva unui polinom este un polinom

Împărțirea a două polinoame f și g înseamnă a determina 2 polinoame q, r astfel încât

a)f=g*q+r; b)grad(r)<grad(g)

În realizarea operațiilor pe polinoame, sunt necesare cunoaștere operațiilor pe monoame.

În ceea ce privește aspectul interfeței calculatorului, interacțiunea cu utilizatorul trebuie sa fie cât mai prietenoasă și intuitivă. De asemenea, utilizatorii trebuie să fie atenționați în momentul în care introduc un polinom invalid. Aceasta parte prezintă o libertate a proiectantului bazată în mare parte pe imaginație.

3.2 Modelare

După determinarea domeniului problemei, datele și procesele sunt translatate prin abstractizare în domeniul soluției ducând la modele de reprezentare, algoritmi și programe.

Folosind modelarea OOP se poate facilita alinierea obiectelor din lumea reală cu obiectele software, deoarece obiectele cu relațiile și interacțiunile din lumea reală relaționează și interacționează similar și într-o aplicație software. Clasele din implementare pot fi inspirate din domeniul problemei (nume similare, relații și responsabilități similare) ceea ce va asigura o mai bună înțelegere a designului și a implementării.

În cadrul sistemului software a calculatorului de polinoame s-a translatat domeniul matematic al problemei, astfel s-au creat clase cu denumiri specifice: Monomial, Polynomial, Operations; s-a păstrat relația matematică descrisă în subcapitolul precedent între polinom și monom: polinomul conține o listă de monoame în modelul soluției; iar operațiile matematice de adunare, scădere, înmulțire etc. au fost descrise în metode/capabilități ce respectă modelul matematic. Translatarea domeniului matematic a problemei nu prezintă dificultăți majore având în vedere relația strânsă între matematică și conceptul de programare.

Denumirea sugestivă a claselor facilitează înțelegerea lor și definirea atributelor corespunzătoare; clasa Monomial conține ca atribute coeficientul și gradul, luând în calcul modelul matematic.

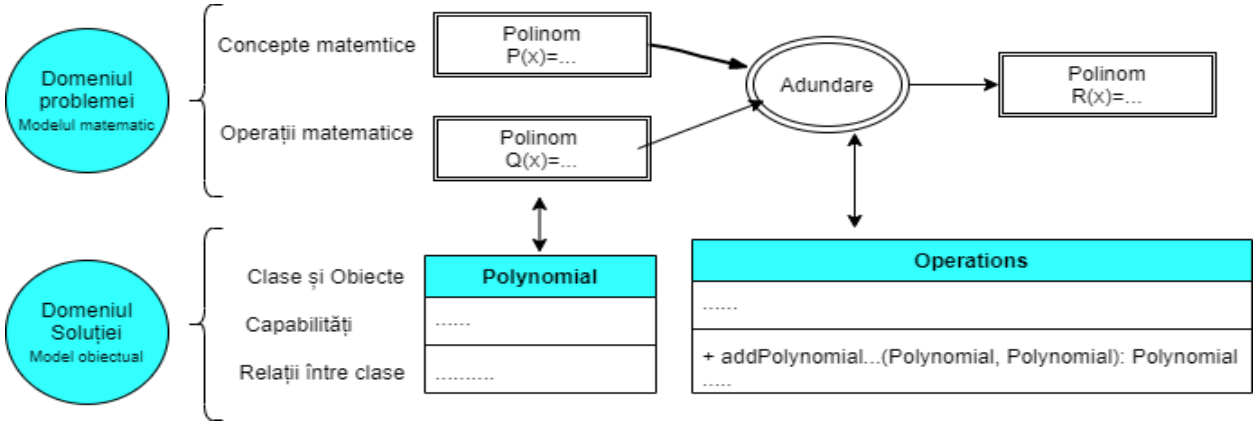

Figure 1 Modelare

3.3 Scenarii și cazuri de utilizare

Identificarea cerințelor și setarea lor clară sunt extrem de importante și trebuie realizate riguros înainte de proiectarea și implementare programului software. Se poate vorbi chiar de o inginerie a cerințelor care se ocupă cu stabilirea serviciilor cerute sistemului și a constrângerilor(acest proces este unul ciclic). În continuare mă voi axa pe două etape ale acestui proces: elicitarea cerințelor și specificarea cerințelor în cadrul temei de laborator.

Elicitarea cerințelor: descoperirea cerințelor sistemului: activități precum interviuri și analiză bazată pe scenarii.

Specificarea cerințelor: implică modelarea riguroasă a cerințelor: use-case, diagrame UML

Din enunțul problemei reies următoarele cerințe pe care trebuie să le îndeplinească sistemul:

a) Existența unei interfețe grafice prin care utilizatorii pot să introducă polinoame
b) Utilizatorii să selecteze operația matematică de executat pe cele două polinoame
c) În interfața grafică să se afișeze rezultatul

Există și constrângeri specificate:

a) Polinoamele introduse să fie de o singură variabilă. Deci, utilizatorii trebuie să introducă polinoame în nederminata X
b) Coeficienții monoamelor să fie numere întregi

Stabilirea unor cerințe și recomandări din cadrul laboratoarelor:

a) Gradele polinoamelor să fie numere pozitive
b) La împărțirea și la integrare polinoamelor, apărând posibilitatea ca polinoamele rezultat să aibă coeficienți de tipul real (double), să se afișeze coeficienții sub această formă.

În următorul tabel sunt definite cerințele sistemului:

Cerințe	
Cerințe funcționale (descriu ce trebuie să facă sistemul)	✓ Calculatorul de polinoame le permite utilizatorilor să insereze polinoame ✓ Calculatorul de polinoame le permite utilizatorilor să selecteze operația matematică. ✓ Calculatorul de polinoame ar trebui să adune, să scadă, înmulțească, să determine derivata sau primitiva polinoamelor introduse, în funcție de ce a selectat utilizatorul. Aceasta se va realiza prin apăsare butonului de egal „=”

	✓ Calculatorul de polinoame trebuie să semnaleze utilizatorul de o în neregulă: introducerea unor polinoame invalide sau împărțirea la 0. ✓ Să permită adăugarea, ștergerea parțială sau totală de conținut (introdus în text-box-urile definite pentru cele două polinoame) cu ajutorul butoanelor
Cerințe non-funcționale (descriu cu ce calitate ar trebui să funcționeze sistemul)	○ Calculatorul de polinoame să fie intuitiv și ușor de folosit ○ Să conțină informații legate de funcționarea sa. ○ Introducerea polinoamelor să se realizeze atât cu ajutorul tastaturii, cât și a butoanelor existente în interfața grafică ○ Semnalizarea unei nereguli să se facă într-un fel cât mai prietenos și ușor de identificat de către utilizator
Constrângeri (descriu limitele de dezvoltare a sistemului)	▪ Operațiile se fac pe polinoame cu coeficienți întregi ▪ Gradul monoamelor trebuie să fie număr natural. ▪ Polinoamele introduse trebuie să fie de o singură variabilă.

Tabel 2 Definirea cerințelor sistemului

Use-case UML capturează grafic actorii sistemului, cazurile de utilizare și relațiile dintre acestea.

Actorii- orice tip de utilizator

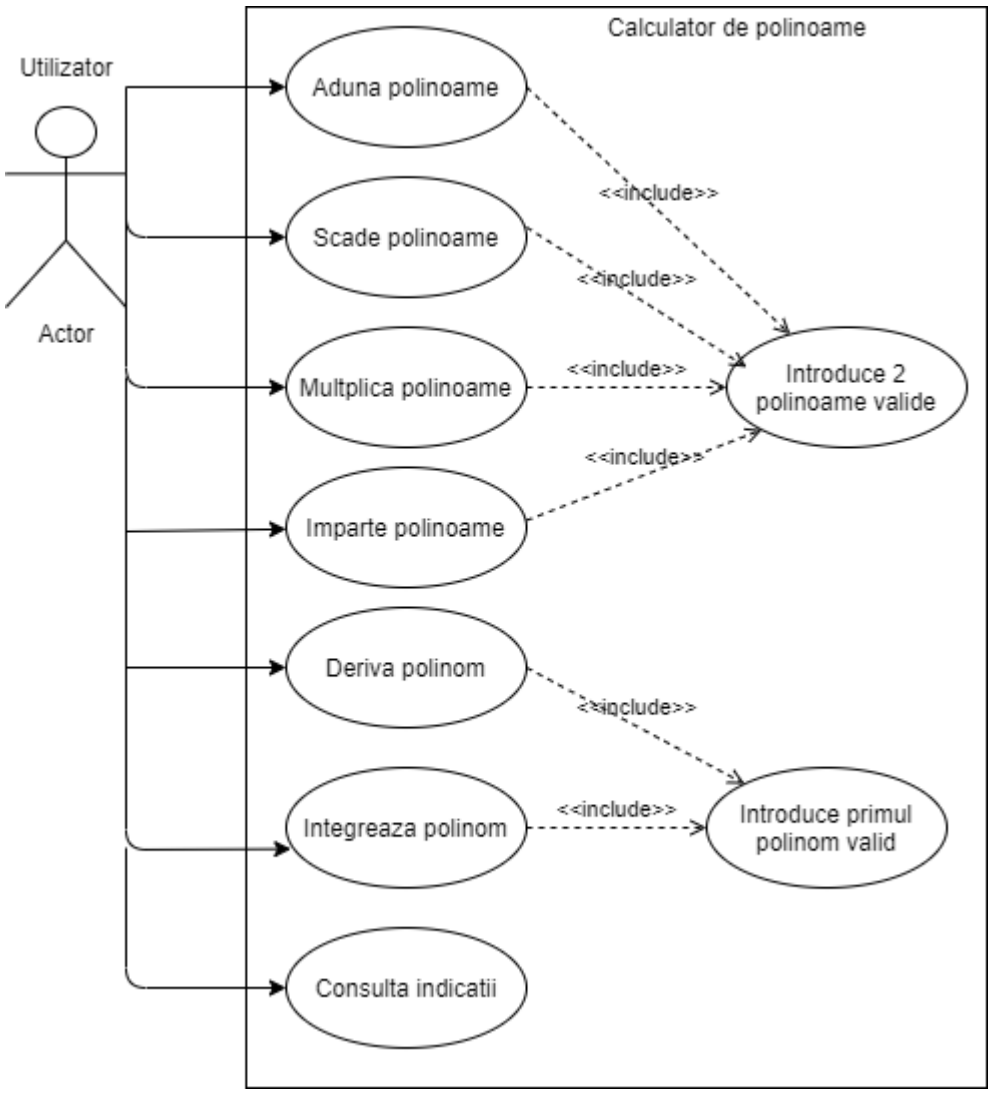

Figure 2 Use-Case UML

Cazurile de utilizare ale aplicației:

Use case: adună polinoame
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul introduce 2 polinoame în text-box-urile corespunzătoare fie cu ajutorul tastelor sau cu ajutorul butoanelor disponibile în interfață.
2. Utilizatorul selectează operația „+ addition" -aceasta se va înroși dacă e selectată
3. Utilizatorul apasă pe butonul de egal „="
4. Calculatorul de polinoame executa operația de adunare a celor două polinoame și afișează rezultatul în text-box-ul rezervat pentru rezultat pe interfață.

Secvențe secundare

- Utilizatorul introduce polinoame invalide (ex. cu coeficienți reali, cu mai multe variabile) sau care nu respectă indicațiile din secțiunea „info" a calculatorului de polinoame. În acest caz, text-boxul unde se află greșeala va fi colorat roșu
- Utilizatorul introduce un număr de mai mult de 9 caractere. Va fi informat printr-un mesaj de warning.
- Scenariul se întoarce la pasul 1 până când text-box-urile vor fi verzi sau utilizatorul respectă indicațiile primite prin warning.

Use case: scade polinoame
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul introduce 2 polinoame în text-box-urile corespunzătoare fie cu ajutorul tastelor sau cu ajutorul butoanelor disponibile în interfață.
2. Utilizatorul selectează operația „- Subtraction" -aceasta se va înroși dacă e selectată
3. Utilizatorul apasă pe butonul de egal „="
4. Calculatorul de polinoame execută operația de scădere a celor două polinoame și afișează rezultatul în text-box-ul rezervat pentru rezultat pe interfață. Din primul polinom se scade al doilea

Secvențe secundare

- Utilizatorul introduce polinoame invalide (ex. cu coeficienți reali, cu mai multe variabile) sau care nu respectă indicațiile din secțiunea „info" a calculatorului de polinoame. În acest caz, text-boxul unde se află greșeala va fi colorat roșu
- Utilizatorul introduce un număr de mai mult de 9 caractere. Va fi informat printr-un mesaj de warning.
- Scenariul se întoarce la pasul 1 până când text-box-urile vor fi verzi sau utilizatorul respectă indicațiile primite prin warning.

Use case: multiplică polinoame
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul introduce 2 polinoame în text-box-urile corespunzătoare fie cu ajutorul tastelor sau cu ajutorul butoanelor disponibile în interfață.
2. Utilizatorul selectează operația „* Multiplication" -aceasta se va înroși dacă e selectată
3. Utilizatorul apasă pe butonul de egal „="
4. Calculatorul de polinoame execută operația de multiplicare a celor două polinoame și afișează rezultatul în text-box-ul rezervat pentru rezultat pe interfață.

Secvențe secundare

- Utilizatorul introduce polinoame invalide (ex. cu coeficienți reali, cu mai multe variabile) sau care nu respectă indicațiile din secțiunea „info" a calculatorului de polinoame. În acest caz, text-boxul unde se află greșeala va fi colorat roșu
- Utilizatorul introduce un număr de mai mult de 9 caractere. Va fi informat printr-un mesaj de warning.
- Scenariul se întoarce la pasul 1 până când text-box-urile vor fi verzi sau utilizatorul respectă indicațiile primite prin warning.

Use case: împarte polinoame
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul introduce 2 polinoame în text-box-urile corespunzătoare fie cu ajutorul tastelor sau cu ajutorul butoanelor disponibile în interfață.
2. Utilizatorul selectează operația „/ Division" -aceasta se va înroși dacă e selectată
3. Utilizatorul apasă pe butonul de egal „="
4. Calculatorul de polinoame execută operația de împărțire a celor două polinoame și afișează rezultatul în interfață. Se vor afișa două polinoame ca rezultat, câtul și restul.

Secvențe secundare

- Utilizatorul introduce polinoame invalide (ex. cu coeficienți reali, cu mai multe variabile) sau care nu respectă indicațiile din secțiunea „info" a calculatorului de polinoame. În acest caz, text-boxul unde se află greșeala va fi colorat roșu.
- Utilizatorul introduce pentru al doilea polinom un polinom nul sau nu scrie nimic. În acest caz, se va afișa un mesaj de warning prin care i se specifică faptul că împărțirea la 0 nu e posibilă.
- Utilizatorul introduce un număr de mai mult de 9 caractere. Va fi informat printr-un mesaj de warning.
- Scenariul se întoarce la pasul 1 până când text-box-urile vor fi verzi sau utilizatorul respectă indicațiile primite prin warning.

Use case: derivă polinom
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul introduce un polinom în text-box-ul corespunzător primului polinom fie cu ajutorul tastelor sau cu ajutorul butoanelor disponibile în interfață.
2. Utilizatorul selectează operația „d Derivative" -aceasta se va înroși dacă e selectată. În momentul apăsării acestei operații text boxul celui de-al doilea polinom va dispărea
3. Utilizatorul apasă pe butonul de egal „="
4. Calculatorul de polinoame execută operația de derivare a primului polinom și afișează rezultatul în text-box-ul rezervat pentru rezultat în interfață.

Secvențe secundare

- Utilizatorul introduce polinom invalid (ex. cu coeficienți reali, cu mai multe variabile) sau care nu respectă indicațiile din secțiunea „info" a calculatorului de polinoame. În acest caz, text-boxul va fi colorat roșu
- Utilizatorul introduce un număr de mai mult de 9 caractere. Va fi informat printr-un mesaj de warning.
- Scenariul se întoarce la pasul 1 până când text-box-ul va fi verde sau utilizatorul respectă indicațiile primite prin warning.

Use case: integrează polinom
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul introduce un polinom în text-box-ul corespunzător primului polinom fie cu ajutorul tastelor sau cu ajutorul butoanelor disponibile în interfață.
2. Utilizatorul selectează operația „∫ Integration" -aceasta se va înroși dacă e selectată
3. Utilizatorul apasă pe butonul de egal „="
4. Calculatorul de polinoame executa operația de integrare a primului polinom și afișează rezultatul în text-box-ul rezervat pentru rezultat în interfață.

Secvențe secundare

- Utilizatorul introduce polinom invalid (ex. cu coeficienți reali, cu mai multe variabile) sau care nu respectă indicațiile din secțiunea „info" a calculatorului de polinoame. În acest caz, text-boxul va fi colorat roșu
- Utilizatorul introduce un număr de mai mult de 9 caractere. Va fi informat printr-un mesaj de warning.
- Scenariul se întoarce la pasul 1 până când text-box-ul va fi verde sau utilizatorul respectă indicațiile primite prin warning.

Use case: consultă indicații
Actor principal: utilizator
Princpalul scenariu de succes:

1. Utilizatorul selectează secțiunea info din cadrul interfeței grafice. Pentru a reveni la calculator selectează „new calculus”

Având clar stabilite cazurile de utilizare, putem trece la parte de proiectare a aplicației.

4. Proiectare

Obiectivul etapei de design sau proiectare este obținerea unui sistem bun prin utilizarea eficientă a unor resurse existente, precum și a unor resurse noi. De asemenea, scopul ei este de a sparge complexitatea folosind fie abordări de tipul top-down sau bttom-up. În realizarea proiectării calculatorului de polinoame am folosit ambele abordări. S-a folosit paradigma OOP.

Putem defini mai multe nivele în cadrul etapei de design:

Design level 1

În cadrul primului nivel regăsim imaginea globală a sistemului. Grafic se reprezintă sub forma unei „cutii negre” asupra căreia se trimit date de intrare și procesele ascunse în interiorul cutiei furnizează un rezultat. Calculatorul de polinoame este un sistem *determinist*, adică pentru aceleași date de intrare tot timpul avem același rezultat pe ieșire.

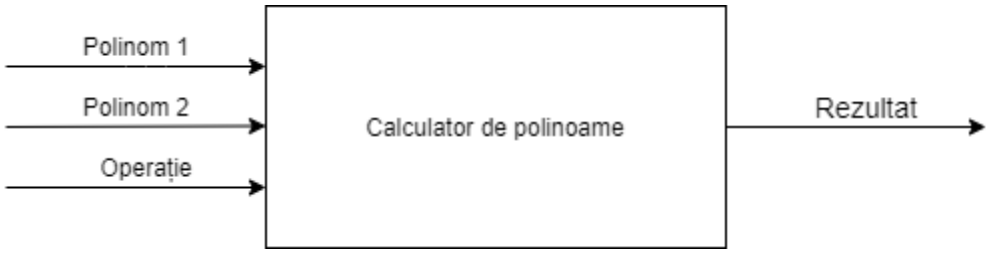

Figure 3 Cutia neagră a sistemului

Design level 2

Sistemul calculatorului de polinoame este divizat în 3 subsisteme care prezintă pattern-ul arhitectural Model View Controller (MVC). Acest pattern arhitectural minimizează dependențele/interacțiunile între subsisteme și se evită relațiile circulare între acestea.

Conforma pattern-ului MVC, modelul are rolul de a încapsula datele și funcționalitățile aplicației. În cadrul acestuia se definesc obiectele, operațiile și se gestionează rezultatele obținute cu datele introduse de utilizatori cu scopul preluării acestora de interfața grafică. Subsistemul View conține clasele care definesc interfața grafică a aplicației, având un rol important în interacțiunea directă cu utilizatorii. Subsistemul Controller are rol decizional, el primește ca intrări evenimente ce au loc la nivelul View-ului (apăsarea unui buton, mișcarea mouse-ului etc.) și le transformă în comenzi fie pentru interfața grafică, fie pentru model.

Cel mai mare câștig al acestui model arhitectural este faptul că a despărțit partea ce ține de interfața grafică de subsistemul ce se ocupă cu partea de calcul a calculatorului de polinoame. Acesta arhitectură oferă posibilitatea reutilizării modelului, precum ușurează testarea și întreținerea codului.

În schema de mai jos este ilustrată împărțirea sistemului „Calculator de polinoame” în cele trei subsisteme menționate și de asemenea se pot vizualiza relațiile dintre acestea.

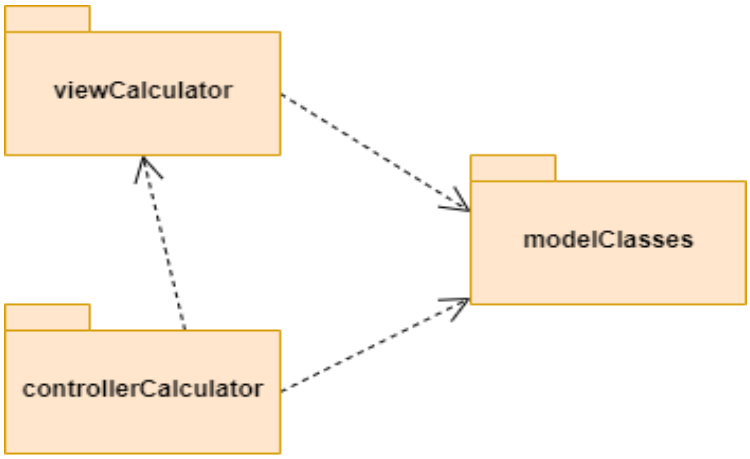

Figure 4 Diagrame pachete. MVC pattern

Subsistemul viewCalculator interacționează cu controllerCalculator prin transmiterea de evenimente (de exemplu apăsarea unui buton, introducerea unor valori de către utilizator). În cadrul claselor din controllerCalculator se tratează răspunsul la eveniment și este transmis înapoi la subsistemul viewCalculator pentru a face modificările corespunzătoare la nivelul interfeței grafice. De asemenea controllerCalculator interacționează cu modelClasses, transmițându-i noi comenzi interpretate de la interfață. modelClasses va actualiza noile date, iar viewCalculator le va prelua cu scopul afișării lor în interfață.

De exemplu: Utilizatorul introduce polinoamele, selectează operația de adunare și apasă butonul de egal. În momentul acesta, viewCalculator trimite un semnal către controllerCalculator; acesta din urmă interpretează faptul că se vrea operația de adunare și transmite subsistemului modelClasses valorile citite din interfață și faptul că se vrea adunarea lor. ModelClasses se ocupă de operația de adunare și setează valoarea rezultatului ca apoi viewController să îl preia și sa îl afișeze în interfața grafică.

În proiect există și pachetul polynomialCalculator care conține o clasa cu metoda main din care se rulează aplicația.

Design level 3

În cadrul acestui nivel se definesc clasele fiecărui subsistem și interacțiunile dintre ele. Denumirea claselor, cum este amintit în al treilea capitol, e sugestivă și inspirată din domeniul problemei.

Apartenența claselor la un subsistem este marcată prin culori diferite. Astfel, pachetul pentru model (mov) conține clasele ce definesc obiectele și operațiile ce se realizează la nivelul datelor cu scopul setării rezultatului: Monomial, Polynomial, Operations etc. . ViewController conține clasele View și InfoPanel responsabile pentru interfața grafică cu utilizatorul, iar în pachetul controllerCalculator e definită clasa ControllerCalc cu clasele sale interne ce gestionează răspunsurile la evenimentele primite de la nivelul interfeței GUI. Mai jos se regăsește diagrama UML a claselor care ilustrează structura logică a sistemului. Funcționalitățile claselor vor fi abordate în detaliu în capitolul 5.

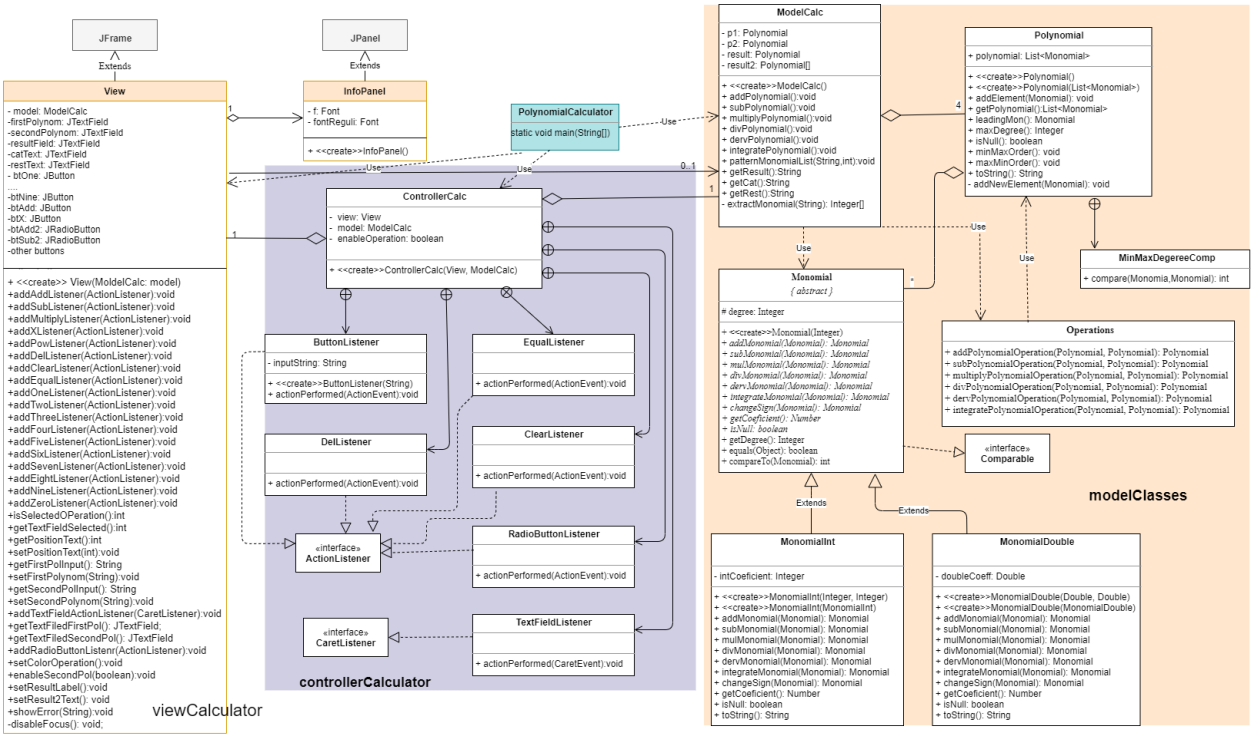

Figure 5 Diagrame UML

Design-ul de nivel 4, care se referă la împărțirea claselor în rutine este ilustrat în cadrul diagramei UML. Există două metode *private*. „extractMonomial" din clasa ModelCalc extrage sub formă de întregi coeficientul și gradul unui monom dintr-un sir de caractere corespunzător formei unui monom. A doua metodă *private* este „addNewElement" din clasa Polynomial care adaugă în lista de monoame a unui polinom un nou element. Dacă există un monom de același grad în listă, atunci se modifică coeficientul acestuia corespunzător. Am folosit această funcție private, pentru că se apelează atât din constructorul care creează un polinom primind o listă de monoame cât și din metoda „addElement" cu ajutorul căreia se poate adăuga oricând un nou element în lista de monoame a polinomului.

Design level 5

În cadrul etapei 5, dintre algoritmii folosiți voi aminti pe cel de împărțire a două polinoame. Celelalte operații de adunare, scădere, înmulțire , derivare și integrare au fost realizate cu operații simple la nivelul monoamelor celor două polinoame. În continuare voi prezenta pașii algoritmului de împărțire, definit în cazul etapei de proiectare și esențial pentru implementarea acestei operații.

Amintim că pentru împărțirea a doua polinoame P1 și P2 trebuie determinate două polinoame Q și R astfel încât P1=P2* Q+ R. Pentru realizarea acestei operații trebuie urmați următorii pași:

a. Se împarte monomul dominant al deîmpărțitului (P1) la monomul dominant al împărțitorului (P2). Se obține astfel monomul dominant al câtului.
b. Se înmulțește monomul obținut la cât cu împărțitorul P2 și produsul obținut se scade din deîmpărțitul P1. Se obține polinomul $P1_1$
c. Se continuă împărțirea luând ca deîmpărțit polinomul $P1_1$ și se împarte monomul dominant al lui $P1_1$ la monomul dominant al lui P2 rezultând al doilea monom al câtului.
d. Se repetă procedeul anterior până când polinomul $P1_x$ are gradul inferior gradului polinomului P2. $P1_x$ va fi restul împărțirii.

Pseudocod-ul algoritmului de împărțire:

```
function n/d is
  require d<>0
  Q<-0
  R<-0
while r<>0 and degree(R)>=degree(d) do
        t<- lead(R)/ lead(d)
        Q<-Q+t;
        R<- R- t*d
return (Q, R)
```

Având partea de proiectare finalizată se poate trece implementarea claselor cu funcționalitățile acestora.

5. Implementare

5.1 Clase și metode

În ceea ce privește denumirea claselor și a metodelor s-au ales nume sugestive care să reflecte domeniul problemei, un exemplu fiind cel prezentat în subcapitol *Modelare*. De asemenea am pus accentul pe proprietatea de reutilizarea a codului, de aceea clasele din modelController, Monomial și Polynomial pot fi folosite independent de celelalte clase ale sistemului în alte programe, câștig obținut datorita arhitecturii MVC. În continuare sunt prezentate clasele din cele 3 pachete ale sistemului împreună cu câmpurile și capabilitățile lor importante.

a. **Clase din pachetul „modelClasses”**

Sistemul, calculatorul de polinoame trebuie să gestioneze calculele cu polinoame după cum reiese din cerințele menționate anterior. Având în vedere faptul că polinomul este considerat ca fiind o listă de monoame, voi începe prin a prezenta clasa care descrie această unitate, monomul.

Clasa abstractă ***Monomial*** -este clasa care descrie modelul matematic de monom. Un monom în matematică se caracterizează prin coeficientul și gradul nedeterminatei X. Deoarece coeficientul poate să fie atât număr întreg, cât și număr real fracționar am decis ca această clasă să fie abstractă și să fie moștenită de două clase care să manipuleze operațiile pe cele două tipuri diferite ale coeficienților: MonomialInt și MonomialDouble. Această clasă este comparabilă deoarece implementează interfața Comparable<Monomial>.

Această clasă abstractă are un singur **atribut** care se referă la gradul monomului.

Atribute	Descriere
protected Integer degree	-acest atribut reprezintă gradul unui monom -acest câmp este de tipul întreg pentru a respecta specificațiile din cerință -vizibilitatea acestuia e protected pentru a putea fi accesat în principal de subclase sau alte clase din pachet

Clasa Monomial are un singur **constructor**:

Constructori	Descriere
public Monomial(Integer degree)	-acest constructor are rolul de a seta atributul degree cu valoarea trimisă ca parametru -nu se poate instanția un obiect cu *new Monomial(...)* deoarece clasa Monomial e abstractă; în acest caz, utilizarea acestui constroctur este esențială în constructorii claselor MonomialInt și MonomialDouble, acesta fiind apelat în prima linie cu *super(degree)*

Metodele care manipulează operațiile matematice pe monoame sunt abstracte. Implementarea lor se va face în cadrul claselor MonomialInt și MonomialDouble respectând convențiile specifice tipului de coeficienți. Operația de scădere nu era neapărat necesară de implementat deoarece în cadrul operațiilor pe polinoame nu mă folosesc de ea, dar având în vedere că *Monomial* face parte din modelClasses am implementat-o pentru proprietatea de reutilizare a codului.

- *public abstract Monomial addMonomial(Monomial m);*
  - metodă abstractă ce trebuie implementă în clasele descendente
  - descrie definiția antetului metodei ce are ca scop implementarea operației de adunare a două monoame de același grad și care returnează un obiect de tipul Monomial
- public abstract Monomial subMonomial(Monomial m);
  - metodă abstractă ce trebuie implementă în clasele descendente
  - descrie definiția antetului metodei ce are ca scop implementarea operației de scădere a două monoame de același grad și care returnează un obiect de tipul Monomial
- *public abstract Monomial mulMonomial(Monomial m);*
  - metodă abstractă ce trebuie implementă în clasele descendente
  - descrie definiția antetului metodei ce are ca scop implementarea operației de înmulțire a două monoame și care returnează un obiect de tipul Monomial
- *public abstract Monomial divMonomial(Monomial m);*
  - metodă abstractă ce trebuie implementă în clasele descendente
  - descrie definiția antetului metodei ce are ca scop implementarea operației de împărțire a două monoame și care returnează un obiect de tipul Monomial
- *public abstract Monomial dervMonomial();*
  - metodă abstractă ce trebuie implementă în clasele descendente
  - descrie definiția antetului metodei ce are ca scop implementarea operației de derivare a monomului și care returnează un obiect de tipul Monomial reprezentând derivata
- *public abstract Monomial integrateMonomial();*
  - metodă abstractă ce trebuie implementă în clasele descendente
  - descrie definiția antetului metodei ce are ca scop implementarea operației de integrare a monomului și care returnează un obiect de tipul Monomial reprezentând primitiva
- *public abstract Monomial changeSign();*
  - metodă abstractă ce trebuie implementă în clasele descendente
  - descrie definiția antetului metodei ce are ca scop implementarea operației de schimbare a semnului coeficientului monomului și care returnează un obiect de tipul Monomial reprezentând rezultatul
- *public abstract Number getCoefficient();*
  - metodă abstractă ce trebuie implementă în clasele descendente
  - va returna un obiect de tipul Number, astfel că în clasele derivate unde e implementată se va returna coeficientul monmomului, fie el număr întreg sau double
- *public abstract boolean isNull();*
  - metodă abstractă ce trebuie implementă în clasele descendente
  - va returna un boolean; true- dacă coeficientul monomului e 0 sau false în caz contrar
- public Integer getDegree()
  - este o metodă de tipul get, care returnează valorea atributului *degree* a monomului.
- (@Override) public boolean equals(Object o);
  - Se suprascrie metoda „equals” din metoda object, astfel încât egalitatea dintre monoame să se fie la nivelul gradului; prin această abordare mi-am simplificat codul de adăugare a unui monom nou în lista de monoame a polinomului
- (@Override) public int compareTo(Monomial o);
  - Se stabilește relația dintre monoame.
  - Un monom cu grad mai mare e „mai mic decât” un monom cu grad mai mic. Astfel în momentul sortării listei de monoame a polinomului, monoamele cu grad mai mare sunt primele.

Clasa **MonomialInt** -este clasa care descrie modelul matematic de monom cu coeficienți întregi. Această clasă moștenește clasa *Monomial* și implementează toate metodele abstracte ale acesteia.

Atribute	Descriere
private Integer intCoeficient	-acest atribut reprezintă coeficientul monomului - este de tipul întreg -vizibilitatea acestuia e private; accesul se face prin metode de tipul get și set

Această clasa are doi constructori:

Constructori	Descriere
public MonomialInt(Integer coefficient, Integer degree)	-prima linie din constructor este apelul constructorului clasei părinte Monomial: super(degree) -după se setează valoare câmpului intCoeficient cu valoare trimisă ca parametru
public MonomialInt(Monomial m)	-creează un nou obiect de tipul MonomialInt cu datele din monomul primit ca parametru

S-au implementat metodele abstracte din clasa *Monomial*. Raționamentul pentru metodele *addMonomial(Monomial m), subMonomial*(*Monomial m*), mulMonomial(*Monomial m*) este similar. Se verifică pesimist de ce tip e monomul trimis ca parametru. Astfel, dacă acest monom este de tipul MonomialDouble, rezultatul operației va fi un monom de tipul Double și se returnează. Dacă monomul primit e de tipul MonomialInt, operațiile vor genera tot coeficienți întregi pentru monomul rezultat, astfel că se creează și returnează un monom de tipul MonomialInt.

Verificare pesimistă s-a realizat cu ajutorul următoarei sintaxe: „*m instanceof MonomialDouble*” -această valoare returnează true dacă monomul m e de tipul MonomialDouble și false în caz contrar.

Pentru operația de adunare și scădere se fac operațiile corespunzătoare pe coeficienții celor două polinoame. Scăderea se realizează dintre monomul curent și cel trimis ca parametru. În cazul în care coeficientul rezultatului e 0, atunci se returnează monomul nul, iar în caz contrar se returnează monomul cu coeficientul rezultat și același grad ca al celor două monoame.

Pentru a verifica dacă un coeficient Double e 0, s-a folosit următoarea expresie: „Math.abs(newCoeff)<= 0.00001”

În cazul înmulțirii, pentru gradul monomului rezultat s-a făcut adunarea gradelor celor doua monoame (cel curent și cel trimis ca parametru). În cazul în care cel puțin un termen al înmulțirii a fost 0, rezultatul este monomul nul.

Pentru metoda de împărțire a monomului curent la un alt monom trimis ca parametru, *divMonomial(Monomial m)* , verific din nou pesimist de ce tip e monomul trimis ca parametru. Am adăugat în definiția metodei „throws ArithmeticException” pentru cazul în care cel de-al doilea monom e nul. În cazul în care monomul trimis ca parametru e de tipul MonomialDouble, atunci monomul rezultat returnat va fi de tipul MonomialDouble. Dacă acesta va fi însă tot de tipul MonomialInt, atunci se verifică dacă restul împărțirii coeficienților e 0 sau nu, prin următoare expresie „this.intCoeficient%(Integer)m.getCoefficient()!=0” (s-a făcut downcasting la tipul Integer pentru rezultatul de tip Number al metodei m.getCoefficient()). În cazul în care restul e 0, atunci monomul rezultat e de tipul MonomialInt, în caz contrar de tipul MonomialDouble. Logica operațiilor ce se fac este la fel ca pentru modelul matematic.

Ce e special în metoda *dervMonomial()* este tratarea cazului în care monomul e constant (are gradul 0) și atunci se returnează monomul nul. Aceasta face derivarea monomului.

În metoda *integrateMonomial()* se decide ce tip de monom să se returneze. Astfel, dacă expresia „this.intCoeficient%(this.degree+1)==0” este adevărată, atunci se returnează un monom de tipul MonomialInt, iar în caz contrar de tipul MonomialDouble.

Metoda *changeSign()* returnează un nou obiect MonomialInt cu coeficientul de semn opus.

Metoda *getCoefficient()* returnează coeficientul monomului.

Este suprascrisă metoda *toString()* pentru a afișa monomul sub forma aX^b, unde a este coeficientul întreg și b gradul. În cadrul acesteia se tratează cazurile în care gradul e 0, coeficientul este 1 sau monomul e nul.

Clasa *MonomialDouble* moștenește clasa abstractă Monomial și reprezintă monomul cu coeficient de tipul double. Implementarea acestei clase este similară cu cea a clasei MonomialInt, astfel pentru evitarea redundanței în explicații voi enunța aspectele importante.

Atribute	Descriere
private Double doubleCoeff	-acest atribut reprezintă coeficientul monomului - este de tipul Double -vizibilitatea acestuia e private; accesul se face prin metoda de tipul get

Și MonomialDouble are doi constuctori:

Constructori	Descriere
public MonomialDouble(Double coefficient, Integer degree)	-prima linie din constructor este apelul constructorului clasei părinte Monomial: super(degree) -după se setează valoare câmpului doubleCoeff cu valoare trimisă ca parametru
public MonomialDouble(Monomial m)	-creează un nou obiect de tipul MonomialDouble cu datele din monomul primit ca parametru

Metodele abstracte din Monomial au fost implementate, iar logica de implementare e similară cu cea de la MonomialInt. Pentru că nu se cere derivarea și integrarea pentru polinoame cu coeficienți de tipul double în cadrul cerinței problemei, metodele *dervMonomial*() și *integrateMonomial*() returnează null, urmând ca acestea să fie implementate în cazul dezvoltării aplicației.

O altă remarcă se referă la funcția *toString()*. Pentru a afișa coeficienții doar cu două zecimale am folosit următorul raționament: „str=Math.round(doubleCoeff*100.0)/100.0”.

Clasa *Polynomial* a fost implementată conform modelului matematic prin care se afirmă faptul că un polinom este format din monoame. În acest caz, un obiect de tipul Polynomial are ca atribut o listă de Monomials. Obiectele de tip Polynomial vor fi folosite pentru operațiile manipulate de calculatorul de polinoame.

Atribute	Descriere
private ArrayList<Monomial> polynomial= new ArrayList<>();	-acest atribut reprezintă lista de monoame a polinomului - este de tipul ArrayList -vizibilitatea acestuia e private; accesul se face prin metoda de tipul get; pentru a adăuga elemente există două funcții speciale

Clasa Polynomial are 2 constructori:

Constructori	Descriere
public Polynomial()	-se adaugă în lista de monoame monomul nul
public Polynomial(ArrayList<Monomial> monomials)	-se primește ca parametru o listă de monoame oarecare (pot exista și duplicate la nivel de grad) -se parcurge lista de monoame primită ca parametru și se adaugă elementele cu ajutorul metodei private *addNewElement(Monomial m)* în lista de monoame a obiectului

Metoda *private void addNewElement(Monomial m)* este cea care se ocupă cu adăugarea unui monom în lista de monoame a obiectului rezolvând cazul în care există deja un monom cu același grad în listă. În acest caz, se poate verifica direct cu expresia „polynomial.contains(m)” dacă există un monom cu același grad (motivul pentru care am suprascris metoda Equals din clasa Monomial). Dacă există, atunci se extrage din listă acel monom, se adună cu

monomul trimis ca parametru, iar monomul rezultat în urma adunării se adaugă în lista de monoame. Se tratează și cazul în care se obține monomul nul, acesta nu se mai adaugă în lista de monoame. La final, se face ordonarea monoamelor în listă în ordine descrescătoare a gradului prin apelul metodei maxMinOrder().

Metoda *public void addElement(final Monomial m)* este metoda prin care se poate adăuga din alte clase un monom în lista de monoame a obiectului. Aceasta garantează prin atributul „final" faptul că se va adăuga o copie a monomului în lista de monoame a polinomului. Această metodă va apela metoda private *addNewElement(Monomial m).*

Metoda *public Monomial leadingMon()* va returna o copie a termenului dominant, adică elementul cu grad maxim.

Metoda *Integer maxDegree()* returnează gradul unui monom. Această metodă apelează metoda precedentă pentru a găsi termenul dominant și returnează gradul acestuia.

Metoda *public boolean isNull()* verifică dacă polinomul e nul, adică termenul său dominant e monomul nul. În caz afirmativ returnează true.

Metoda *public void minMaxOrder()* ordonează monoamele polinomului în ordine crescătoare a gradelor în listă. E folositoare pentru derivarea polinomului. În această metodă se apelează „Collections.sort(polynomial,new MinMaxDegreeComp());" . De aceea mi-am definit o subclasă comparator.

Metoda *public void maxMinOrder()* ordonează monoamele polinomului în ordine descrescătoare a gradelor în listă. E folositoare pentru integrarea polinomului și pentru afișarea lui. Se apelează direct „Collections.sort(polynomial);" deoarece clasa Monomial e comparabilă fiindcă implementează interfața Comparable.

E suprascrisă metoda *public String toString()* astfel încât polinomul să fie afișat sub formă de String ușor de citit.

Clasa internă *MinMaxDegreeComp* implementează interfața Comparator<Monomial>. Este suprascrisă metoda *public int compare(Monomial o1, Monomial o2)* astfel încât monomul cu grad mai mic să fie înaintea monomului cu grad mai mare în relația de dordine. Este necesară pentru metoda *minMaxOrder()* din clasa Polynomial.

Clasa *Operations* este o clasă care conține metode statice ce au ca parametri două polinoame și returnează un polinom ce reprezintă rezultatul operației descrise. Având în vedere proprietățile polinoamelor menționate în capitolul 3, toate cele 6 operații pe polinoame vor returna un polinom nou ca rezultat. Lista de parametri a metodelor conține atributul final, ceea ce îi dă garanția utilizatorului că polinoamele transmise nu vor fi modificate în interiorul metodei.

- public static Polynomial addPolynomialOperation(final Polynomial p1, final Polynomial p2)
  - reprezintă operația de adunare a polinoamelor p1 și p2
  - rezultatul va fi tot un polinom
  - se face o copie a primului polinom și se adaugă apoi în acea copie, cu ajutorul funcției *addElement(Monomial m),* monoame din cel de-al doilea polinom. Funcționalitatea metodei din clasa Polynomial a fost explicată anterior, iar ce este important este faptul că în interiorul ei se folosește adunare definită pentru clasa Monomial.
- public static Polynomial subPolynomialOperation(final Polynomial p1, final Polynomial p2)
  - reprezintă operația de scădere a polinoamelor p1 și p2 (p1-p2)
  - rezultatul va fi tot un polinom
  - se face o copie a primului polinom și se adaugă apoi în acea copie, cu ajutorul funcției addElement(Monomial m), monoame din cel de-al doilea polinom dar cum semnul opus al coeficientului (se apelează metoda *changeSign()).*
- public static Polynomial multiplyPolynomialOperation(final Polynomial p1, final Polynomial p2)
  - reprezintă operația de înmulțire a polinoamelor p1 și p2
  - rezultatul va fi un polinom
  - se creează un polinom cu *new Polynomial()* și se parcurg listele celor două polinoame cu 2 for-uri imbricate, astfel încât fiecare monom din lista lui P1 să se înmulțească cu fiecare monom din lista lui P2. Se folosește următoarea sintaxă : *result.addElement(mon1.mulMonomial(mon2))* , unde se observă că se apelează la operația de înmulțire a monoamelor.
  - pentru parcurgerea listei de monoame s-a folosit *foreach* în loc de *for(int i=0 ...)*
- public static Polynomial[] divPolynomialOperation(final Polynomial p1, final Polynomial p2) throws NullPointerException, ArithmeticException
  - reprezintă operația de împărțire a două polinoame P1/P2

- rezultatul este un vector de două polinoame, astfel încât pe prima poziție să fie inserat câtul împărțirii, iar pe a doua poziție se va afla restul
- metoda poate arunca două excepții, dacă s-a transmis un polinom neinițializat sau dacă cel de-al doilea polinom este nul (conține doar monomul nul), astfel că împărțirea nu are sens
- implementarea metodei se bazează pe algoritmul de împărțire descris anterior
- se folosesc metodele *isNull()* și *maxDegree()* definite în clasa Polynomial, pentru a verifica dacă polinomul e nul (conține numai monomul nul) și pentru a afla gradul unui polinom, funcții ce sunt necesare pentru formarea condiției buclei *while*
- se folosesc metodele de adunare, scădere și înmulțire din clasa Operations definite anterior pentru calculul noului rest. În cazul fiecărei iterații se adaugă în polinomul pentru cât rezultatul împărțirii monomului dominant al deîmpărțitului (de fapt a restului care se inițializează la început cu valorile lui P1) la monomul dominant al lui P2.

- public static Polynomial dervPolynomialOperation(final Polynomial p)
  - reprezintă operația de derivare a unui polinom
  - rezultatul e tot un polinom
  - se face o copie a polinomului și se ordonează monoamele din listă în ordinea crescătoare a gradelor
  - se parcuge lista de monoame, se derivează monomul curent cu metoda specifică a obiectului Monomial și monomul obținut se adaugă în polinomul rezultat
- public static Polynomial integratePolynomialOperation(final Polynomial p)
  - e similară cu metoda de derivare numai că se face ordonarea în ordinea descrescătoare a gradelor și se face apelul metodei de integrare a monoamelor din listă

Clasa *ModelCalc* este responsabilă pentru a manipula rezultatul operațiilor ce se cer de utilizator prin interfața grafică și pregătește răspunsul pentru a fi afișat.

Atribute	Descriere
private Polynomial p1;	-atribut private ce se setează prin intermediul unei metode descrise mai jos -acesta va fi setat cu primul polinom introdus de utilizator
private Polynomial p2;	-atribut private ce se setează prin intermediul unei metode descrise mai jos -acesta va fi setat cu al doilea polinom introdus de utilizator
private Polynomial result;	- atribut private ce se setatează la instanțierea unui obiect de tipul ModelCalc cu polinomul nul -acesta va fi setat cu rezultatul operației selectate de utilizator pe polinoamele p1 și p2 -va fi accesat cu metoda publica *String getResult()*
private Polynomial[] result2= new Polynomial[2];	- reprezintă un vector de polinoame folosit pentru rezultatul operației de împărțire; astfel pe prima poziție se va afla câtul, iar pe a doua poziție restul -acesta va fi setat cu rezultatul operației de împărțire dintre p1 și p2 -accesul se face prin doua metode de tipul *get* (accesoare) *String getCat()* și *String getRest()*

Clasa are un singur constructor:

Constructori	Descriere
public ModelCalc()	-acesta va seta câmpul *result* ca fiind polinomul nul

Metodele publice addPolynomial(), subPolynomial(), multiplyPolynomial(), dervPolynomial() integratePolynomial() vor seta câmpul *result* cu rezultatul operației corespunzătoare. La nivelul acestor metode se va

face apelul metodei statice specifice din clasa *Operations*. Exemplu, în addPolynomial() avem următorul apel „this.result= Operations.addPolynomialOperation(p1,p2);".

Antetul metodei de împărțire e: *public void divPolynomial() throws Exception.* Aceasta poate arunca mai departe excepțiile prinse din apelul metodei statice de împărțire din clasa Operations. Excepțiile prinse vor fi aruncate cu mesaje specifice („Impartirea la 0 nu e posibila"). În cadrul acestei metode se va setat câmpul *result2.*

Metoda publică *patternMonomialList(String s, int choicePol) throws NumberFormatException* va extrage dintr-un șir monoamele și le va adăuga în lista polinomului p1 dacă valoarea lui choicePol e 1 și în p2 pentru alte valori ale lui choicePol.

- Pentru a extrage monoamele dintr-un șir am folosit metodele din pachetul *java.util.regex (matcher(String), find(), group());*
- Mi-am declarat un șir de caractere *patternString* în care se descrie formatul posibil pentru un monom. Am folosit grupuri independente pentru formatele posibile cu relația de „sau" între ele. Un grup independent are forma: (?:.....).
- Se obține obiectul de tipul Pattern prin compilarea string-ului definit anterior: *pattern=Pattern.compile(patternString).*
- Se extrag pe rând grupurile ce reprezintă un monom și se apelează metoda privată *Integer[] extractMonomial(String s) throws NumberFormatException* care returnează 2 valori ce reprezintă coeficientul și gradul polinomului; aceasta aruncă o excepție de tipul *NumberFormatException* dacă numărul de caractere pentru un întreg e mai mare de 9
- Se adaugă monoamele extrase în lista polinomului corespunzător

Metoda privată *Integer[] extractMonomial(String s) throws NumberFormatException* va primi ca parametru un string sub forma unui monom.

- Am folosit metoda *split* pe stringul dat ca parametru și am folosit separatorii posibili tot cu ajutorul grupurilor indepemdente => ((?:*[Xx]\^)|(?:*[Xx])|(?:[Xx]\^)|(?:[Xx])) .
- De exemplu grupul evidențiat se referă la faptul că *X^ e un separator pentru șirul dat. În acest caz se pot extrage ca stringuri separate numărul ce reprezintă coeficientul și numărul ce reprezintă gradul.
- S-a folosit metoda statică Integer.paseInt(...)

b. Pachetul viewCalculator

Acesta conține două clase View și InfoPanel. Aceste clase vor descrie interfața grafică cu utilizatorul.

Clasa View moștenește clasa JFrame. Această clasă are o mulțime de atribute ce reprezintă butoanele calculatorului, etichetele și atribute pentru text-field-uri. Denumirea atributelor e sugestivă. Printre atributele ce țin exclusiv de interfața grafică se numără și atributul *model* de tipul ModelCalc.

Atribute importante	Descriere
Private ModelCalc model	-atribut care se referă la modelul de date cu care va interacționa interfața grafică pentru a afișa rezultatele operațiilor
private JTextField firstPlynom/ secondPolynom/ resultField/ catText/ restText	-sunt atribute ce descriu text-box-urile unde se introduc polinoamele și unde se afișează rezultatele
private final JRadioButton btAdd2/ btMultipy2/ btSub2/ btDiv2/ btDerivate2/ btIntegration 2	-butoane de tipul JRadio folosite pentru selecția operației matematice de executat
private final JButton btZero/btOne/btTwo/ .../btNine	-butoane ce reprezintă cifre pentru ca utilizatorul să scrie polinoamele
private final JButton btResult	-buton care apăsat va determina efectuarea operației dacă se poate realiza sau va genera scrierea unui mesaj de informare
Private final JButton btAdd/ btSub/btPow/ btX/ btDel/ btC	-alte butoane care ajută la scrierea polinoamelor

Constructorul acestei clase va seta atributul model cu cel transmis ca parametru.

Constructori	Descriere
public View (ModelCalc model)	-va seta câmpul *model* cu valoarea paramtrului, astfel se va face legătura între interfața grafică și modelul de date în vederea manipulării operațiilor cerute de utilizatori -în cadrul acestuia se va crea aspectul interfeței grafice (descrisă mai jos), adăugând câmpurile definite anterior

Sunt mai multe metode implementate în cadrul acestei clase. Sunt metode cu denumirea *add...Listener (actionListener e)* prin care se adaugă ascultători butoanelor din cadrul interfeței grafice, astfel încât prin apăsarea lor să se întâmple logica descrisă în *controllerCalc*.

Există o metodă *private void disableFocus()* prin care se dezactivează *focusul* în momentul când se apasă un buton, astfel cursorul să rămână în text-field-ul unde era.

Celelalte metode sunt publice.

Metode importante	Descriere
int isSelectedOperation()	-returnează numărul operației matematice selectate: Addition (0), Subtraction(1)...Integration(5)
int getPositionText/ void setPositionText	-folosite pentru obținerea și setarea poziției cursorului -utilizate pentru operația de ștergere, adăugare a unui caracter cu ajutorul butoanelor
String getFirstPolInput()/ void setFirstPolynom(String s)	-pentru obținerea și setarea textului din text-field-ul primului polinom -metode asemănătoare și pentru cel de-al doilea polinom
addTextFieldsActionListener (CaretListener e)	-se adaugă listener-i pentru text-box-uri astfel încât să se poată realiza verificarea textului introdus de utilizator în timp ce acesta scrie -căsuțele unde se vor scrie cele două polinoame vor fi verzi sau roșii în funcție de corectitudinea textului introdus
void setColorOperation()	-colorează textul operației matematice selectate
void showError(String msg)	-metodă prin care se afișează un mesaj de informare -se face apelul metodei statice *JOptionPane.showMessageDialog(..)*

Clasa InfoPanel moștenește clasa JPanel. Prin intermediul ei s-a creat un obiect JPanel cu informații legate de utilizarea calculatorului de polinoame și a formatului de scriere a monoamelor. Acest obiect a fost adăugat interfeței grafice în interiorul constructorului clasei View. Integrarea lui se va evidenția în subcapitolul dedicat Interfeței Grafice.

c. Pachetul controllerCalculator

Acest pachet conține clasa ControllerCalc care conține mai multe clase interne.

Clasa ControllerCalc are rolul de a gestiona evenimentele ce se petrec la nivelul interfeței grafice și îi oferă un răspuns.

Atribute	Descriere
private View view;	-atribut private ce reprezintă interfața grafică pe care o controlează
private ModelCalc model	-atribut private ce reprezintă modelul de date ce trebuie actualizat conform cerințelor date de utilizator prin intermediul interfeței grafice -cu ajutorul modelului, controllerul oferă un răspuns evenimentelor primite de la interfața grafică

private boolean enableOperation=false;	- atribut boolean private ce se inițializează cu valoare false -manipularea acestui atribut va determina dacă pentru datele introduse (polinoame) se poate da un rezultat în momentul apăsării butonului „=” de către utilizator

Această clasă are un singur constructor:

Constructori	Descriere
public ControllerCalc(View view, ModelCalc model)	-acesta va seta câmpurile *view și model* ce reprezintă interfața grafică și modelul pe care obiectul de tip ControllerCalc le va controla -în interiorul constructului se apelează metodele din *view* care setează ActionListener pentru butoanele interfeței și textfield-urile acesteia. Apelul metodelor se face cu un obiect de tipul unei clase interne (specifice evenimentului) ce implementează interfața ActionListener.

Clasele interne ale clasei ControllerCalc implementează interfața ActionLisner și au rol în definirea Listeneri-lor pentru butoanele și textfield-urile din interfața grafică și implementarea răspunsurilor evenimentelor specifice.

Clasa internă	Descriere
ButtonListener	-implementează interfața ActionListener -acest listener este adăugat butoanelor folosite pentru a scrie un polinom -atribute: private String inputString -constructor: public ButtonListener(String s) • Se setează campul *inputString* cu stringul s primit ca parametru • Acesta de fapt reprezintă valoarea de pe butonul ce este apăsat -metoda implementată: *public void actionPerformed(ActionEvent e)* • Aceasta are rolul de a adăuga valorea *inputString* în momentul în care se apasă un buton, în text field-ul polinomului unde se află cursorul • Pe scurt, această metodă preia textul din căsuța selectată, află poziția cursorului, inserează valoarea butonului apăsat după cursor și setează poziția acestuia după caracterul inserat • Se folosește metoda *substring* pe obiectul de tip String a despărți textul în două, se adaugă caracterul și apoi se face concatenarea
DelListener	-implementează interfața ActionListener -nu are atribute -implementează metoda *actionPerformed* cu scopul operației inverse din cazul clasei anterioare. La apăsare butonului „Del” se șterge caracterul de dinaintea cursorului
ClearListener	-implementează interfața ActionListener -nu are atribute - implementează metoda *actionPerformed* cu scopul operației de ștergere a textului din text-field-ul în care se află cursorul la apăsare butonului „C”
EqualListener	-implementează interfața ActionListener -nu are atribute - implementează metoda *actionPerformed* pentru apăsarea butonului „=”. • Dacă atributul *enableOperation* al controllerului este true, atunci se poate realiza operația selectată de utilizator pentru polinoamele din cele două text-box-uri, în caz contrar se va afișa un mesaj de informare. • Se actualizează câmpurile p1 și p2 din model cu polinoamele introduse de utilizator • În funcție de operația selectată, controllerul transmite modelului să performeze operația specifică și să actualizeze rezultatul • În cazul operației de împărțire s-a folosit blocul try-catch. În cazul unei erori, controllerul va transmite interfeței mesajul de informare pentru a fi afișat

	• În final (în cazul succesului) i se cere interfeței grafice să preia rezultatul din model
TextFieldListener	-implementează interfața CaretListener -nu are atribute -implementează metoda *caretUpdate(CarteEvent e)* pentru validare polinoamelor în timp ce utilizatorul le introduce. Am folosit funcții din pachetul java.util.regex. În stringul pentru pattern am pus mai multe grupuri neacceptate. De fiecare dată când utilizatorul scrie un caracter, listener-ul se activează și verifică dacă există un grup neacceptat de caractere. În cazul în care există, text-box-ul se înroșește și *enableOperation* devine false. Dacă nu există astfel de grupuri, text-boxul este verde și se poate poate efectua operația matematică selectată.
RadioButtonListener	-implementează interfața ActionListener -nu are atribute - implementează metoda *actionPerformed* pentru apăsarea butoanelor JRadio cu ajutorul cărora se selectează operația matematică • Daca operație e derivare sau integrare, text-boxul celui de-al doilea polinom va dispărea (se și șterge conținutul acesteia) • Se înroșește textul operației selectate

Clasa *PolynomialCalculator* din pachetul polynomialCalculator conține metoda statică *main*, metodă care pornește aplicația.

5.2 Interfața grafică cu utilizatorul (GUI)

Interfața grafică cu utilizatorul este destul de intuitivă și folosește elemente din pachetul *javax.swing*. Am încercat să ofer un aspect cât mai prietenos pentru utilizator și am optat de-ai oferi posibilitatea de a adăuga polinoamele și cu ajutorul mouse-ului folosindu-se de butoanele disponibile. De asemenea, utilizatorul poate accesa secțiunea *Info* unde sunt menționate informațiile necesare utilizării calculatorului de polinoame. Odată cu introducerea de caractere, se verifică validitatea datelor introduse. Astfel, dacă se introduce ceva greșit, text-box-ul se va înroși. Selectarea operației de executat se realizează prin apăsarea butonului „=”, iar dacă s-a întâlnit o eroare în execuția operației, utilizatorul va fi informat printr-un mesaj.

Pentru a crea două panouri, cel dedicat calculelor („New Calculus”) și Info, am folosit JTabbedPane care permite selecția panoului de afișat. Pentru Info am instanțiat în constuctorul clasei View un obiect de tipul InfoPanel.

Panoul pentru „New Calculus” este format dintr-un panou principal (*mainPanel*) căruia i s-au adăugat subPanouri. Layout-ul definit pentru acest panou este BoxLayout pe axa Oy. În figura de mai jos e prezentată aranjarea subpanourilor panoului mainPanel.

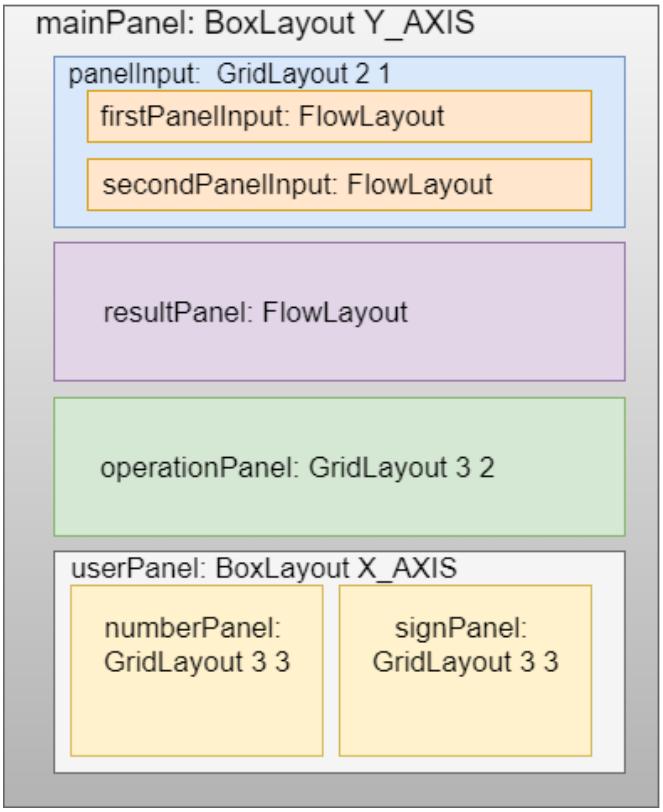

Figure 6 Reprezentare mainPanel

Butoanele, etichetele și text-boxurile au fost adăugate în panourile specifice. Dimensiunea minimă a calculatorului (care e și optimă) a fost setată 750x900.

Câteva imagini cu aspectul calculatorului:

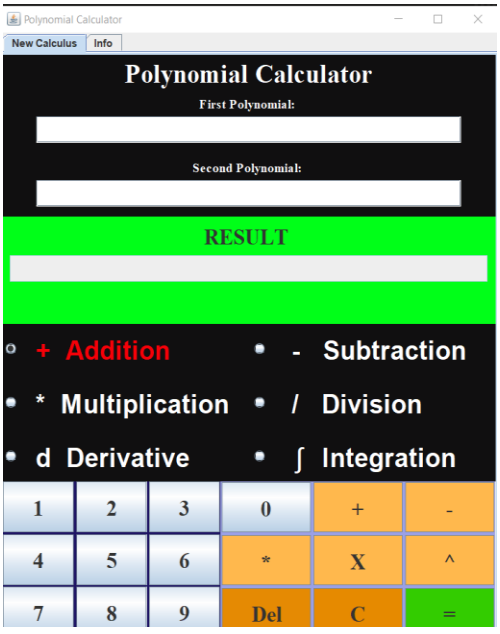

Figure 7 La pornirea aplicației

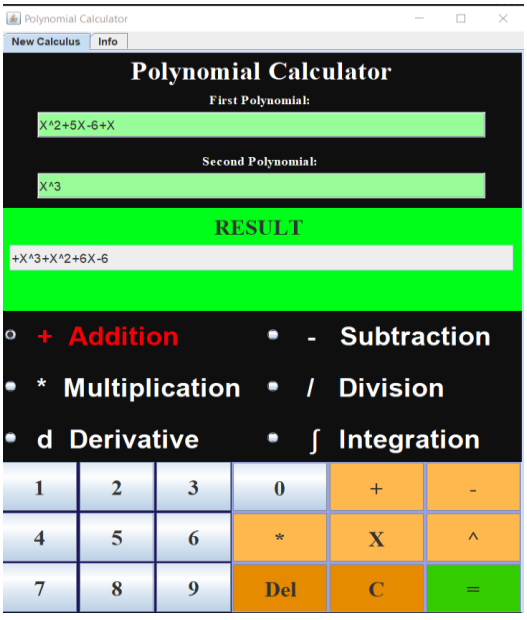

Figure 8 Realizarea opearției de adunare

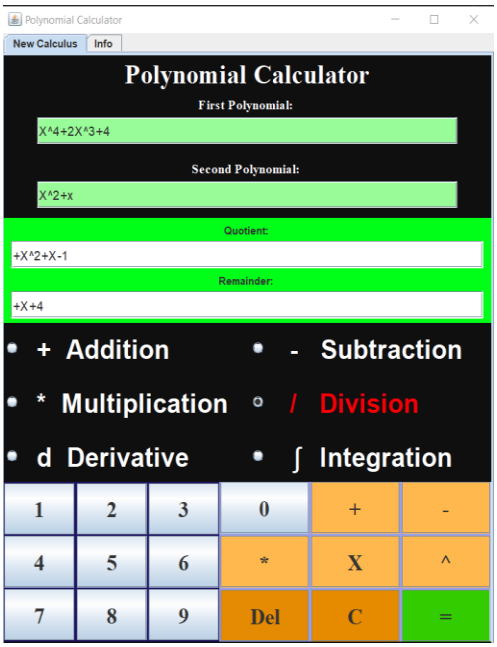

Figure 9 Realizarea operației de împărțire

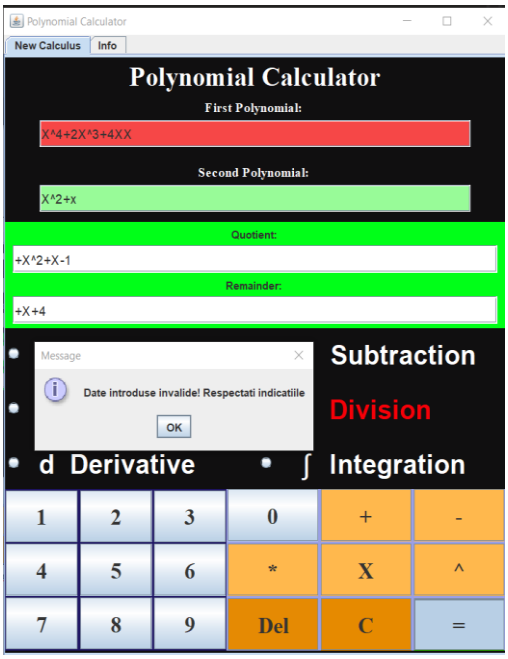

Figure 10 Inrtroducerea de date invalide

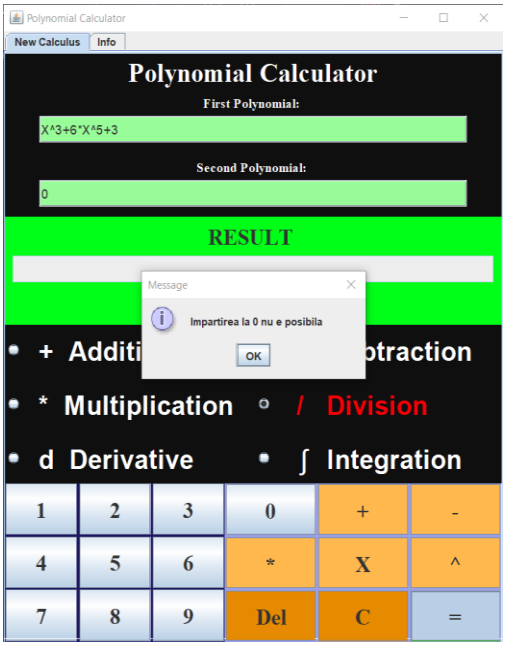

Figure 11 Impărțirea la 0

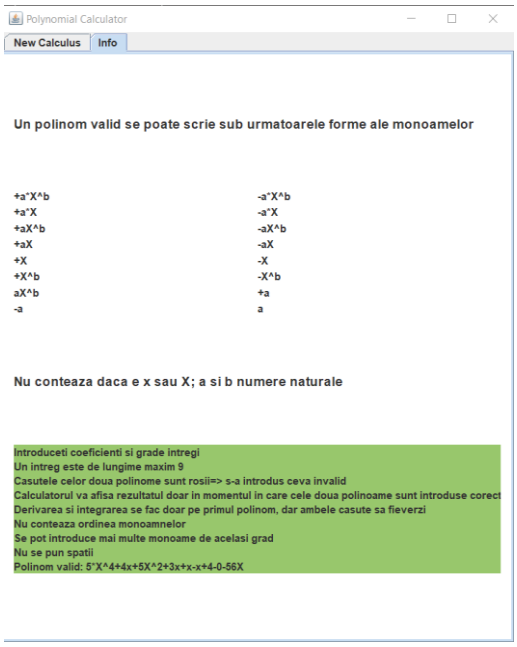

Figure 12 Informații despre utilizare

6. Rezultate

Pentru verificarea corectitudinii funcționalității modelului am realizat mai multe teste folosind JUnit. Am folosit metoda de testare cu parametri, astfel încât pentru o operație s-a repetat testul cu cel puțin cinci argumente diferite.

În primă fază s-a testat funcționalitatea operațiilor pe monoame, operații folosite implicit pentru operațiile ce se realizează pe polinoame. Parametrii transmiși au fost de tipul Monomial pentru operanzi, iar pentru rezultatele așteptate s-au transmis coeficientul și gradul. Am încercat să cuprind cazurile speciale (înmulțirea cu 0, obținerea monomului nul prin scăderea a două monoame egale etc.).

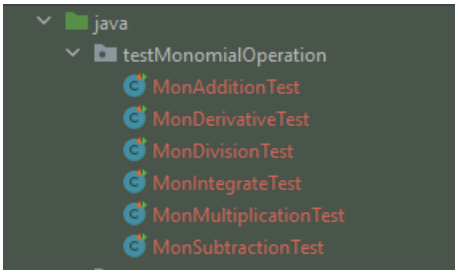

Figure 13 Teste operații monoame

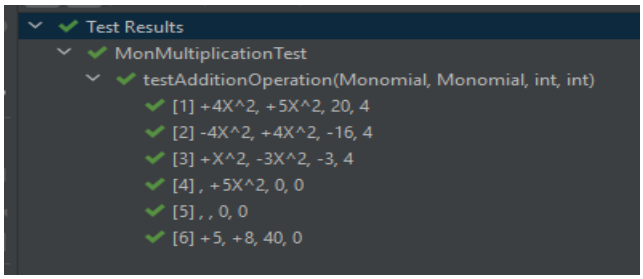

Figure 14 Testele pentru operția de înmulțire monoame

Pentru a verifica atât corectitudinea modelului și implicit și a operațiilor pe polinoame, am instanțiat un obiect de tipul ModelCalc. Parametrii transmiși ca argumente au fost sub formă de string-uri, pentru a verifica și metodele toString ale claselor MonomialInt, MonomialDouble și Polynomial, cu tratarea cazurilor speciale (gradul 0 sau coeficient +1 /-1 etc.). De asemenea, se verifică implicit și extragerea monoamelor cu ajutorul funcțiilor din pachetul *regex*. S-au realizat cel puțin 10 teste pentru fiecare operație.

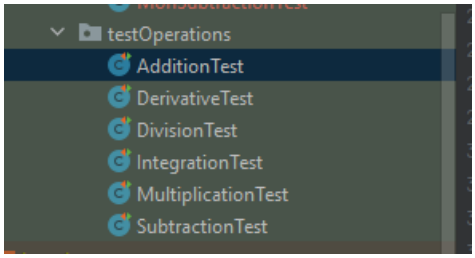

Figure 15 Teste operții polinoame

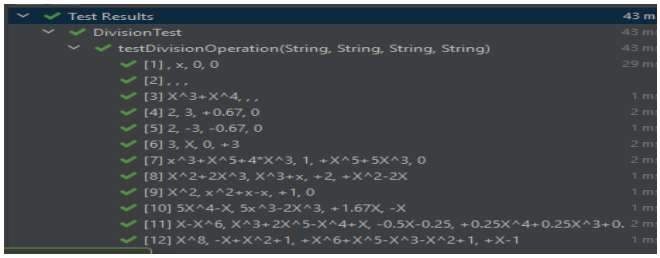

Figure 16 Testare operație împărțire polinoame

7. Concluzii

În concluzie, prin realizarea acestei teme de laborator, s-a rezolvat problema efectuării diferitelor operații pe polinoame care necesitau timp și atenție deosebită dacă se realizau pe foaie. Calculatorul, având o interfață prietenoasă și intuitivă, poate fi folosit de orice utilizator, iar rezultatul poate fi citit și interpretat foarte ușor deoarece este afișat în ordinea descrescătoare a gradelor monoamelor și fără duplicate.

Pentru proiectarea aplicației s-a folosit paradigma orientată pe obiect, folosindu-se pattern-ul arhitectural MVC, care îi oferă proprietate de reutilizare. Câștigul pe care îl aduce acest model arhitectural este faptul că va despărți implementarea modelului de date de interfața grafică și logica de control. În aceste condiții, întreținerea codului este mai ușor de realizat, iar modelul poate fi reutilizat de alte aplicații. De asemenea și dezvoltarea aplicației de față se poate realiza independent, fie la nivelul interfeței grafice, fie în cadrul modelului.

Pentru cel care a realizat aplicația, dezvoltarea ei l-a familiarizat mai mult cu conceptul de Object Oriented și cu modul de modelare, proiectare și implementare a unei aplicații folosind această paradigmă de programare. De asemenea, l-a obișnuit cu urmarea pașilor logici în procesul de dezvoltare a unei aplicații software , de la analiza

problemei, fixarea cerințelor, proiectarea, implementarea și testarea ei. A învățat să manipuleze un șir de caractere și să extragă bucăți din acesta folosindu-se de funcții din pachetul *java.regex*. De asemenea, a exersat modul de testare cu JUnit a aplicației folosind varianta cu parametri.

Ca posibilă dezvoltare ar fi adăugarea unui istoric al operațiilor astfel încât utilizatorul să realizeze operații înlănțuite. De asemenea, se poate implementa și logica pentru polinoame cu coeficienți reali și se pot adăuga operații noi cum ar fi calcularea funcției asociate polinomul sau dacă polinomul introdus e reductibil. O altă dezvoltare, care ar necesita dezvoltarea modelului de date ar fi introducerea polinoamelor cu 2 sau 3 variabile.

8. Bibliografie

- Marius Burtea, Georgeta Burtea, Matematică M1 : trunchi comun şi curriculum diferenţiat : clasa a XII-a , Piteşti: Carminis Educaţional, 2007
- „ASSIGNMENT 1 SUPPORT PRESENTATION” , material auxiliar (furnizat pe platforma Microsoft Teams)
- Cursuri de la disciplina Tehnici de Programare Fundamentală, studiată în semestrul al 2-lea al anului universitar 2020-2021
- https://www.baeldung.com/parameterized-tests-junit-5
- https://docs.oracle.com/javase/tutorial/essential/regex/index.html
- https://docs.oracle.com/javase/7/docs/api/