**PROIECT “POLYNOMIAL PROCESSING”**

Student

FILIP LIVIU PAUL

**Conținut:**

1. Obiectivul temei
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare
4. Implementare
5. Rezultate
6. Concluzii
7. Bibliografie

**Capitolul I. *Obiectivul temei***

Obiectivul temei “Polynomial Processing” îl reprezintă proiectarea și implementarea unei aplicații care este capabilă să gestioneze evenimentele computaționale necesare procesării diferitelor operații matematice efectuate asupra polinoamelor și construirea unei soluții corecte și valide.

Îndeplinirea obiectivului propus necesită completarea unor obiective secundare și pași intermediari, elemente pe care le vom sintetiza în următorul tabel:

|  |  |
| --- | --- |
| Obiectiv secundar | Descriere |
| Înțelegerea conceptelor matematice din teoria polinoamelor, algebră și analiză matematică, necesare pentru implementarea software a operațiilor asupra polinoamelor | * Se prezintă noțiunile de bază despre polinoame, operațiile de adunare, scădere, înmulțire, împărțire a două polinoame, cât și derivarea și integrarea unui polinom * Aceste noțiuni vor fi descrise pe larg în capitolul II. |
| Analizarea problemei, modelarea conceptuală a elementelor ierarhice care constituiesc soluția problemei | * Se vor analiza cerințele problemei, se vor explica acțiunile care trebuie implementate. * Se va explica modelarea soluției pe baza cazurilor de utilizare și a ipotezei problemei. * Aceste sarcini vor fi tratate în capitolul II. |
| Identificarea scenariilor și cazurilor de utilizare a aplicației și rezolvarea eventualelor probleme care pot apărea în urma interacțiunii dintre utilizator și aplicație | * Se vor prezenta cazurile de utilizare sub forma de diagrame și descrieri de use-case. * Descrierile use-case-urilor se vor face sub forma unei liste conținând pașii execușiei pentru fiecare use-case * Se vor identifica eventualele nereguli ce pot apărea la interacțiunea utilizator – program și se va oferi tehnica abordată pentru soluționarea problemei. * Descrierea acestor pași se va face in capitolul II. |
| Alegerea elementelor optime de proiectare corespunzătoare paradigmei de programare orientată pe obiecte  Construirea unei interfețe utilizator prin care interacțiunea dintre sistem și user să fie ușurată, să conțină toate funcționalitățile dorite și să ofere rezultate care pot fi refolosite ulterior | * Se vor expune deciziile de proiectare luate pentru optimizarea aplicației în domeniile timp și memorie. * Se va prezenta proiectarea aplicației în conformitate cu paradigma de programare orientată pe obiecte. * Se prezintă diagramele UML de clase și de pachete pentru proiect. * Se evidențiază structurile de date folosite, algoritmii utilizați și interfețele adăugate. * Se evidențiază structura ierarhică a proiectului, acesta fiind împărțit în pachete, clase și subclase. Relațiile dintre acestea vor fi de asemnea vizualizate cu ajutorul diagramelor UML. * Se va descrie modul de proiectare a interfeței utilizator pentru ca această să satisfacă cerințele utilizatorului, să aibă un aspect “prietenos” și ofere posibilitatea utilizatorului să folosească funcțiile implementate. * Aceste idei vor fi pe larg prezentate în capitolul III. |
| Evidențierea și descrierea etapelor de implementare a problemei. | * Se va prezenta implementarea claselor cu câmpurile și metodele cele mai semnificative. * Se vor evidenția elementele de implementare a unor algoritmi specifici. * Se va descrie modul de implementare a interfeței grafice. * Aceste descrieri vor avea loc în capitolul IV. |
| Interpretarea și evaluarea rezultatelor | * Se vor construi date de intrare, cu ajutorul cărora se vor efectua operațiile implementate în aplicație * Se vor construi atât date valide, cât și invalide pentru evidențierea funcționării aplicației. * Se vor prezenta scenariile pentru testare cu Junit * Aceste procese se vor prezenta în capitolul V. |
| Concluzii cu privire la temă și dezvoltări ulterioare | * Se vor trage concluzii cu privire la tema abordată, aplicabilitatea ei și eventuale dezvoltări ulterioare. * Acest subiect va fi tratat în capitolul VI. |

**Capitolul II. *Analiza problemei, modelare, scenarii, cazuri de utilizare***

Cerința problemei în formă inițială este urmăroarea: “Propuneți, proiectați și implementați un sistem de procesara a polinoamelor de o singură variabilă cu coeficienți întregi”. În următoarele rânduri, vom explica pe scurt noțiunile matematice de care avem nevoie pentru rezolvarea cerinței.

Se consideră (A, +, · ) un inel cu 0 elementul nul si 1 elementul unitate. **Definiție:** ***f =*** , se numeste forma algebrica cu coeficienți în inelul A, a polinomului f, in nedeterminata X. Multimea polinoamelor cu coeficienti in inelul A, in nedeterminata X, se noteaza A[X].

A poate fi Z, Q, R, C sau Zp , p numar natural, p > 1. [1]

Polinoamele sunt construite din [termeni](https://ro.wikipedia.org/w/index.php?title=Termen_(matematic%C4%83)&action=edit&redlink=1) numiți monoame, care sunt alcătuite dintr-o constantă (numită coeficient) înmulțită cu una sau mai multe variabile. [2]

Operații cu polinoame:

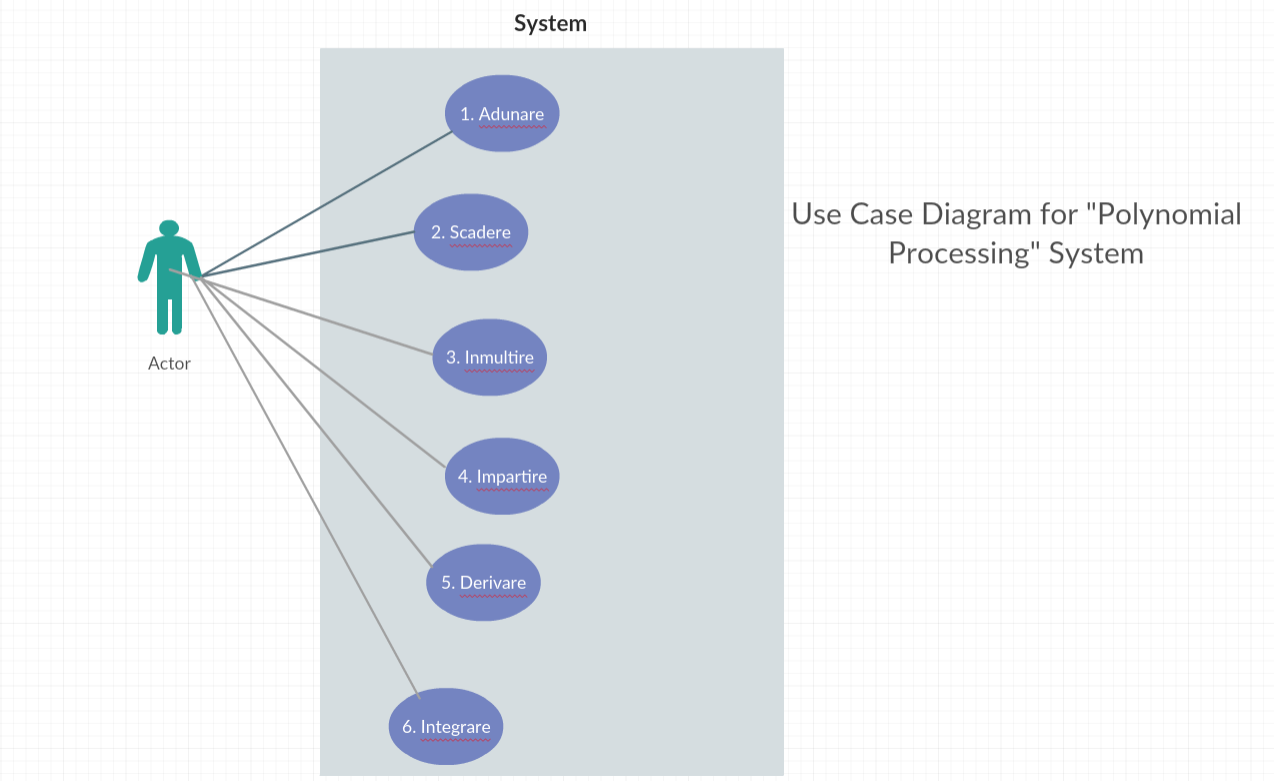
1. **Adunarea polinoamelor:** se face însumând coeficienții monoamelor de același grad.
2. **Scăderea polinoamelor:** se face scăzând coeficienții monoamelor de același grad.
3. **Înmulțirea polinoamelor:** se face înmulțind fiecare monom al primului polinom cu fiecare monom al celui de-al doilea polinom după regula
4. **Împărțirea polinoamelor:** Se utilizează algoritmul de împărțire a polinoamelor:
   * + 1. Se ordonează polinoamele f și g după puterile descrescătoare ale nedeterminatei X
       2. Se face împărțirea polinomului de grad mai mare la polinomul de grad mai mic
       3. Se împarte primul termen al lui f șa primul termen al lui g și se obține astfel primul termen al câtului.
       4. Se înmulțește rezultatul astfel obșinuit cu împărțitorul g și se scade acest produs din deîmpărțitul f. Acest calcul ne dă primul rest al împărțirii
       5. Se repetă procedeul luând primul rest ca deîmpărțit
       6. Algoritmul se termină când gradul restului este mai mic decât gradul împărțitorului.
5. **Derivarea unui polinom:** Se derivează fiecare monom al polinomului, după regula .
6. **Integrarea unui polinom:** Se integrează fiecare monom al polinomului, după regula .

Așadar, operațiile matematice asupra polinoamelor de care avem nevoie sunt **adunarea, scăderea, înmulțirea, împărțirea, derivarea și integrarea.** Toate aceste operații matematice vor fi implementate în aplicația descrisă ulterior.

Modelarea soluției problemei se bazează pe regulile matematice descrise anterior, ținând cont de excepțiile ce pot apărea și cazurile nefavorabile.

Prezentăm în continuare un caz de utilizare sub formă de diagramă use case:

Rolul actorului este luat de către utilizatorul aplicației, acesta poate fi un student, un profesor sau orice altă persoană care dorește să efectueze operații matematice asupra a două polinoame. În cele ce urmează, vom descrie fiecare use case sub forma unei liste conținând pașii execuției fiecărui use case.



1. *Adunarea a două polinoame*

Use case: Adunare

Primary actor: Client

Main succes scenario:

* Clientul introduce primul polinom in zona aferentă
* Sistemul verifică dacă datele introduse sunt corecte
* Clientul introduce al doilea polinom în zona rezervată celui de-al doilea polinom
* Sistemul verifică dacă datele introduse sunt corecte
* Clientul alege operația de adunare a două polinoame
* Sistemul efectuează operația selectată
* În zona rezervată rezultatului este afișat polinomul rezultat din suma celor două polinoame introduse anterior.

Alternative sequences:

1. Unul sau ambele câmpuri pentru introducerea polinoamelor sunt completate cu caractere ilegale pentru descrierea matematică a unui polinom

* În acest caz, sistemul emite un mesaj de eroare care notifică existența caracterelor ilegale în introducerea polinoamelor
* Clientul este întors la pasul de introducere a polinoamelor.

1. Unul din câmpurile pentru introducerea polinoamelor este lăsat necompletat

* În acest caz, sistemul interpretează polinomul ca un polinom nul, deci va efectua adunarea între polinomul introdus corect și 0, rezultatul fiind polinomul introdus corect.

1. *Scăderea a două polinoame*

Use case: Scădere

Primary actor: Client

Main succes scenario:

* Clientul introduce primul polinom in zona aferentă
* Sistemul verifică dacă datele introduse sunt corecte
* Clientul introduce al doilea polinom în zona rezervată celui de-al doilea polinom
* Sistemul verifică dacă datele introduse sunt corecte
* Clientul alege operația de scădere a două polinoame
* Sistemul efectuează operația selectată
* În zona rezervată rezultatului este afișat polinomul rezultat prin diferența dintre polinomul introdus în primul câmp și polinomul introdus în al doilea câmp.

Alternative sequences:

1. Unul sau ambele câmpuri pentru introducerea polinoamelor sunt completate cu caractere ilegale pentru descrierea matematică a unui polinom

* În acest caz, sistemul emite un mesaj de eroare care notifică existența caracterelor ilegale în introducerea polinoamelor
* Clientul este întors la pasul de introducere a polinoamelor.

1. Unul din câmpurile pentru introducerea polinoamelor este lăsat necompletat

* În acest caz, sistemul interpretează polinomul ca un polinom nul, deci va efectua scăderea dintre polinomul introdus în primul câmp și polinomul introdus în al doilea câmp.

1. *Înmulțirea a două polinoame*

Use case: Înmulțire

Primary actor: Client

Main succes scenario:

* Clientul introduce primul polinom in zona aferentă
* Sistemul verifică dacă datele introduse sunt corecte
* Clientul introduce al doilea polinom în zona rezervată celui de-al doilea polinom
* Sistemul verifică dacă datele introduse sunt corecte
* Clientul alege operația de înmulțire a două polinoame
* Sistemul efectuează operația selectată
* În zona rezervată rezultatului este afișat polinomul rezultat prin înmulțirea dintre polinomul introdus în primul câmp și polinomul introdus în al doilea câmp.

Alternative sequences:

1. Unul sau ambele câmpuri pentru introducerea polinoamelor sunt completate cu caractere ilegale pentru descrierea matematică a unui polinom

* În acest caz, sistemul emite un mesaj de eroare care notifică existența caracterelor ilegale în introducerea polinoamelor
* Clientul este întors la pasul de introducere a polinoamelor.

1. Unul din câmpurile pentru introducerea polinoamelor este lăsat necompletat

* În acest caz, sistemul interpretează polinomul ca un polinom nul, deci va realiza înmulțirea dintre 0 și alt polinom ce va duce la un rezultat nul.

1. *Împărțirea a două polinoame*

Use case: Împărțire

Primary actor: Client

Main succes scenario:

* Clientul introduce primul polinom in zona aferentă
* Sistemul verifică dacă datele introduse sunt corecte
* Clientul introduce al doilea polinom în zona rezervată celui de-al doilea polinom
* Sistemul verifică dacă datele introduse sunt corecte
* Clientul alege operația de împărțire a două polinoame
* Sistemul efectuează operația selectată
* În zona rezervată rezultatului este afișat polinomul rezultat prin împărțirea dintre polinomul introdus în primul câmp și polinomul introdus în al doilea câmp.

Alternative sequences:

1. Unul sau ambele câmpuri pentru introducerea polinoamelor sunt completate cu caractere ilegale pentru descrierea matematică a unui polinom

* În acest caz, sistemul emite un mesaj de eroare care notifică existența caracterelor ilegale în introducerea polinoamelor
* Clientul este întors la pasul de introducere a polinoamelor.

1. Unul din câmpurile pentru introducerea polinoamelor este lăsat necompletat

* În acest caz, sistemul interpretează polinomul ca un polinom nul, în cazul în care primul câmp este lăsat necompletat, atunci rezultatul final va fi polinomul nul, iar în cazul în care al doilea câmp este lăsat necompletat, atunci va intra în cazul în care se împarte la 0.

1. Împărțitorul este polinomul nul

* În acest caz, sistemul va semnala o eroare, împărțirea neputându-se face

1. Împărțitorul este un polinom de ordinul I.

* În acest caz, se face împărțirea coeficienților deîmpărțitului la valoarea împărțitorului.

1. *Derivarea unui polinom*

Use case: Derivare

Primary actor: Client

Main succes scenario:

* Clientul introduce primul polinom in zona aferentă
* Sistemul verifică dacă datele introduse sunt corecte
* Clientul alege operația de derivare a unui polinom
* Sistemul efectuează operația selectată
* În zona rezervată rezultatului este afișat polinomul rezultat prin derivare, după regula enunțată

Alternative sequences:

1. Câmpul pentru introducerea polinomului este completat cu caractere ilegale pentru descrierea matematică a unui polinom

* În acest caz, sistemul emite un mesaj de eroare care notifică existența caracterelor ilegale în introducerea polinoamelor
* Clientul este întors la pasul de introducere a polinomului.

1. Câmpul pentru introducerea polinomului este lăsat necompletat

* În acest caz, sistemul interpretează polinomul ca un polinom nul, deci va realiza derivarea lui 0, ce va aduce ca rezultat tot un polinom nul

1. *Integrarea unui polinom*

Use case: Integrare

Primary actor: Client

Main succes scenario:

* Clientul introduce primul polinom in zona aferentă
* Sistemul verifică dacă datele introduse sunt corecte
* Clientul alege operația de derivare a unui polinom
* Sistemul efectuează operația selectată
* În zona rezervată rezultatului este afișat polinomul rezultat prin integrare, după regula enunțată

Alternative sequences:

1. Câmpul pentru introducerea polinomului este completat cu caractere ilegale pentru descrierea matematică a unui polinom

* În acest caz, sistemul emite un mesaj de eroare care notifică existența caracterelor ilegale în introducerea polinoamelor
* Clientul este întors la pasul de introducere a polinomului.

1. Câmpul pentru introducerea polinomului este lăsat necompletat

* În acest caz, sistemul interpretează polinomul ca un polinom nul, deci va realiza derivarea lui 0, ce va aduce ca rezultat tot un polinom nul

**Capitolul III. *Proiectare***

În acest capitol, vom prezenta pe larg deciziile de proiectare luate în concordanță cu paradigma programării orientate pe obiecte, vom expune diagramele UML de clase, structura ierarhică a proiectului și strucuturile de date folosite.

Aplicația este structurată în pachete, cele principale fiind pachetele Model, View și Controller în care sunt implementate atât structurile de date în care stocăm informația despre monoame, respective polinoame, cât și în care avem implementată interfața grafică. Pe lângă acestea, mai avem un pachet intitulat “PT2019.Tema1”, practic clasa App de unde lansăm aplicația și o clasă prin care realizăm testarea unitară.

Diagrama UML pentru pachete este următoarea:

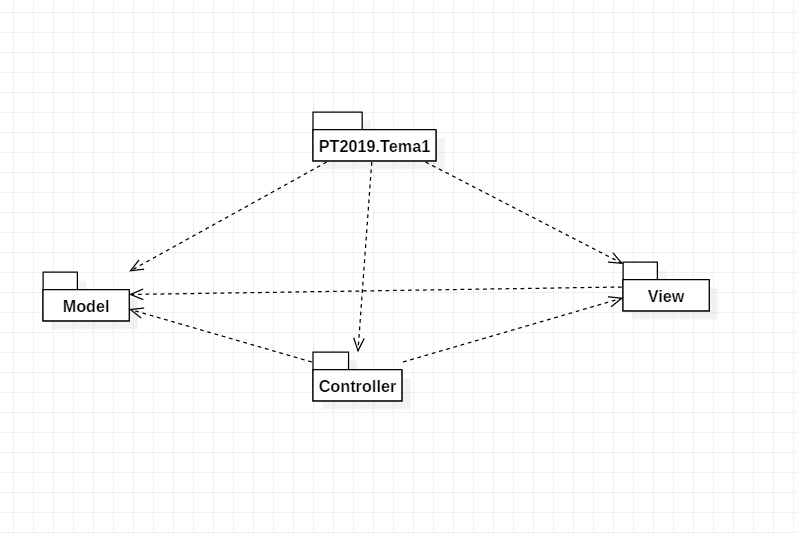


Diagrama UML pentru clasele din pachetul Model:

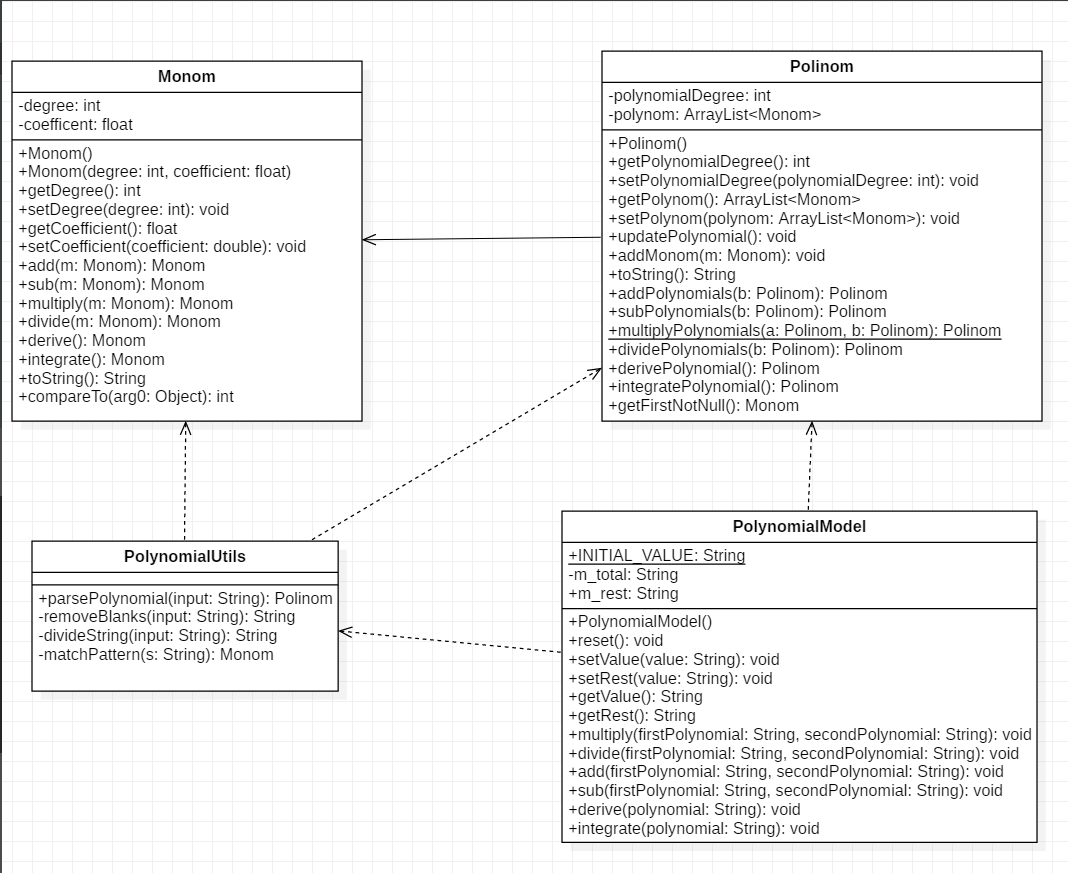


Diagrama UML pentru clasele din pachetul View:

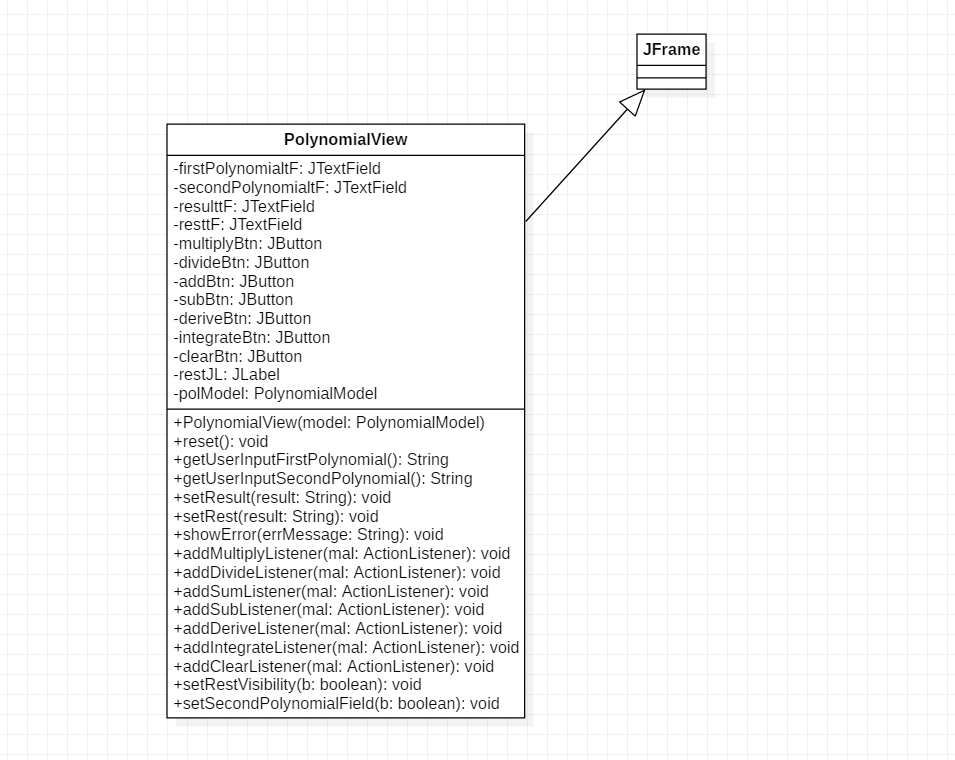


Diagrama UML pentru clasele din pachetul Controller:

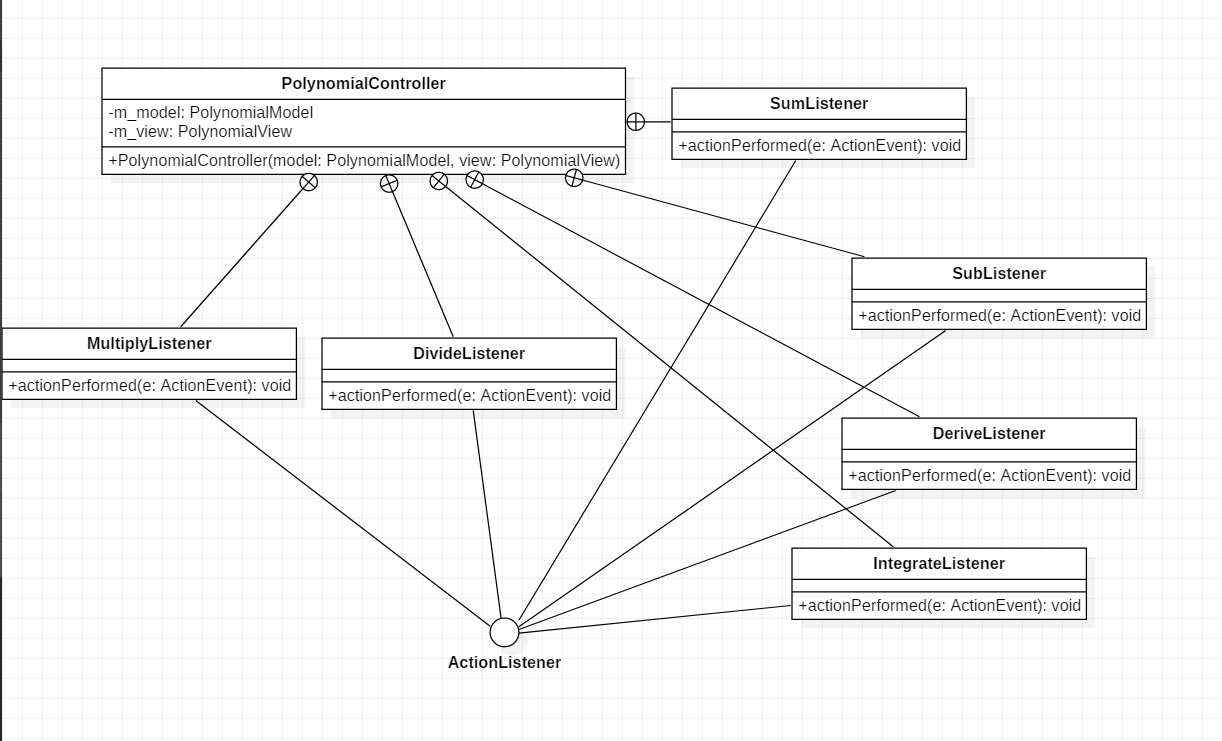
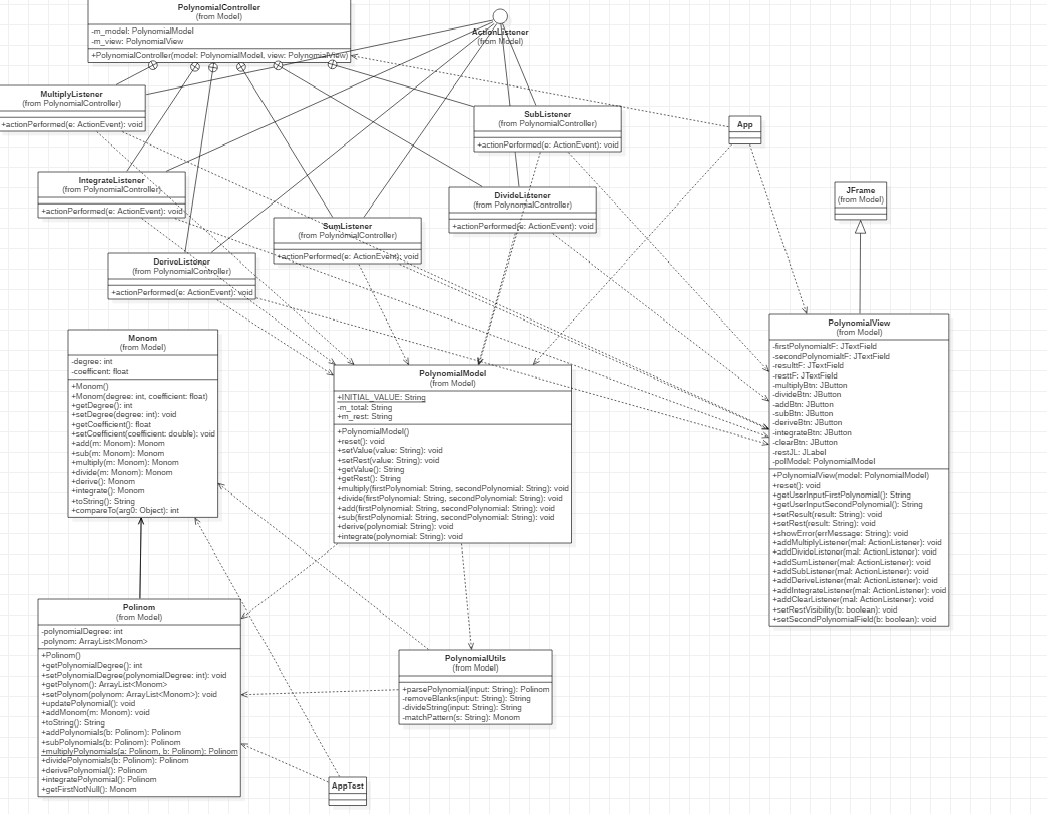


Diagrama UML pentru întreaga aplicație:



Structura principală de date folosită pentru stocarea polinomului va fi o listă de monoame, monom care la rândul lui este definit într-o clasă separată, cu atributele și metodele specifice. Pentru clasa Polinom, vom implementa operațiile cerute, utilizând de asemnea metodele implementate anterior pentru monoame.

Singurul algoritm „mai sofisticat” pe care l-am implementat este cel de împărțire a două polinoame, în conformitate cu ceea ce am explicat în capitolul II.

**Capitolul IV.  *Implementare***

Întrucât aplicația implementată oferă și o interfață grafică, am structurat clasele în pachetele principale după metoda Model-View-Controller. Pachetele construite sunt Model, View, Controller si PT2019.Tema1 (în interiorul căruia lansăm aplicația).

În pachetul Model, am implementat clasele Monom, Polinom, PolynomialUtils și PolynomialModel.

Clasa ***Monom*** are atribute pentru gradul monomului care este un întreg și coeficientul monomului, care este de tip **float**. Metodele implementate în această clasă sunt **add, sub, multiply, divide, derive, integrate, compareTo și toString.**

Metoda **add(Monom m)** primește ca argument un alt monom(de același grad ca insatnța) și returnează un monom nou cu coeficientul format din suma coeficienților monomului instanță și monomului parametru, iar gradul este același. Codul este:

**public** Monom add(Monom m) {

**return** **new** Monom(m.getDegree(), **this**.getCoefficient() + m.getCoefficient());

}

Metoda **sub(Monom m)** primește ca argument un alt monom(de același grad ca insatnța) și returnează un monom nou cu coeficientul format din diferența coeficienților monomului instanță și monomului parametru, iar gradul este același. Codul este:

**public** Monom sub(Monom m) {

**return** **new** Monom(m.getDegree(), **this**.coefficient - m.getCoefficient());

}

Metoda **multiply(Monom m)** primește ca argument un alt monom și returnează un monom nou cu coeficientul format prin înmulțirea coeficienților monomului instanță și monomului parametru, iar gradul este suma coeficienților celor două monoame. Codul este:

**public** Monom multiply(Monom m) {

**return** **new** Monom(**this**.getDegree() + m.getDegree(), **this**.getCoefficient() \* m.getCoefficient());

}

Metoda **divide(Monom m)** primește ca argument un alt monom și returnează un monom nou cu coeficientul format prin împărțirea coeficienților monomului instanță și monomului parametru, iar gradul este diferența coeficienților celor două monoame. Codul este:

**public** Monom divide(Monom m) {

**return** **new** Monom(**this**.getDegree() - m.getDegree(), **this**.getCoefficient() / m.getCoefficient());

}

Metoda **derive()** returnează un nou monom corespunzător derivatei monomului instanță care apelează metoda. Noul coeficient este coeficentul instanței înmulțit cu gradul monomului instanță, iar gradul este cu 1 mai mic decât gradul monomului inițial. În cazul în care gradul monomului este 0, se va returna un monom nul. Codul este:

**public** Monom derive() {

**if** (**this**.degree == 0)

**return** **new** Monom();

**return** **new** Monom(**this**.getDegree() - 1, **this**.getCoefficient() \* **this**.getDegree());

}

Metoda **integrate()** returnează un nou monom corespunzător valorii integrate a monomului instanță care apelează metoda. Noul coeficient este coeficentul instanței împărțit cu gradul monomului instanță +1 , iar gradul este cu 1 mai mare decât gradul monomului inițial. În cazul în care gradul monomului este 0, se va returna un monom nul. Codul este:

**public** Monom integrate() {

**return** **new** Monom(**this**.getDegree() + 1, **this**.getCoefficient() / ((**double**) **this**.getDegree() + 1));

}

Metoda **compareTo(Object arg0)** returnează un întreg, care poate avea valoarea -1, 1 sau 0. În această metoda, vom implementa compararea a două monoame după gradul lor. Metoda este suprascrisă față de metoda inițială din interfața Comparable.

**public** **int** compareTo(Object arg0) {

Monom m = (Monom) arg0;

**if** (**this**.getDegree() == m.getDegree())

**return** 0;

**else** **if** (**this**.getDegree() > m.getDegree())

**return** -1;

**else**

**return** 1;

}

Metoda **toString()** este un override al metodei clasice prin care un monom este transpus sub formă de String pentru o lizibilitate mai mărită.

**public** String toString() {

String m = **new** String();

**if** (**this**.getCoefficient() == 0)

**return** "";

**else** {

**if** (**this**.getDegree() == 0) {

**if** (**this**.getCoefficient() < 0)

m += **this**.getCoefficient();

**else**

m = "+" + **this**.getCoefficient();

} **else** {

**if** (**this**.getDegree() == 1) {

**if** (**this**.getCoefficient() == 1) {

m = "+x";

} **else** {

**if** (**this**.getCoefficient() == -1)

m = "-x";

**else** {

**if** (**this**.getCoefficient() < 0)

m = **this**.getCoefficient() + "x";

**else**

m = "+" + **this**.getCoefficient() + "x";

}

}

} **else** {

**if** (**this**.getCoefficient() == 1) {

m = "+x^" + **this**.getDegree();

} **else** {

**if** (**this**.getCoefficient() == -1) {

m = "-x^" + **this**.getDegree();

} **else** {

**if** (**this**.getCoefficient() < 0)

m = **this**.getCoefficient() + "x^" + **this**.getDegree();

**else**

m = "+" + **this**.getCoefficient() + "x^" + **this**.getDegree();

}

}

}

}

**return** m;

}

}

Clasa ***Polinom*** are atribute pentru gradul polinomului care este un întreg și o listă de monoame, care este de tip **ArrayList<Monom>**. Metodele implementate în această clasă sunt **updatePolynomial, addMonom, addPolynomials, subPolynomials, multiplyPolynomials, dividePolynomials, derivePolynomial, integratePolynomial, getFirstNotNull și toString.**

Metoda **updatePolynomial()** actualizează gradul unui polinom în cazul în care după anumite operații, gradul acestuia scade.

Metoda **addMonom(Monom m)** adaugă un monom transmis prin parametru la lista de monoame a polinomului instanță. Metoda adaugă monomul în ordinea descrescătoare a gradelor monoamelor din lista de monoame.

Metoda **addPolynomials(Polinom b)** realizează suma dintre polinoame și returnează polinomul rezultat.

Metoda **subPolynomials(Polinom b)** realizează diferență dintre polinoame și returnează polinomul rezultat.

Metoda **multiplyPolynomials(Polinom a, Polinom b)** realizează înmulțirea dintre polinoame și returnează polinomul rezultat.

Metoda **dividePolynomials(Polinom b)** realizează impărțirea dintre polinoame și returnează polinomul cât, iar valoarea instanței după apelul metodei va fi chiar restul împărțirii.

Metoda **derivePolynomial()** realizează derivarea polinomului și returnează polinomul rezultat.

Metoda **integratePolynomial()** realizează integrarea polinomului și returnează polinomul rezultat.

Metoda **getFirstNotNull()** returnează primul monom cu coeficient nenul din lista de monoame a polinomului instanță.

Metoda **toString()** este un override al metodei clasice prin care un polinom este transpus sub formă de String pentru o lizibilitate mai mărită.

Codul pentru clasa Polinom este următorul

**package** Model;

**import** java.util.ArrayList;

**public** **class** Polinom {

**private** ArrayList<Monom> polynom;

**private** **int** polynomialDegree;

**public** ArrayList<Monom> getPolynom() {

**return** polynom;

}

**public** **void** setPolynom(ArrayList<Monom> polynom) {

**this**.polynom = polynom;

}

**public** **int** getPolynomialDegree() {

**return** polynomialDegree;

}

**public** **void** setPolynomialDegree(**int** polynomialDegree) {

**this**.polynomialDegree = polynomialDegree;

}

**public** Polinom() {

**super**();

**this**.polynom = **new** ArrayList<Monom>();

**this**.polynomialDegree = 0;

}

**public** **void** updatePolynomial() {

**int** grad = 0;

**for** (Monom a : **this**.getPolynom()) {

**if** (a.getCoefficient() != 0) {

**if** (a.getDegree() > grad)

grad = a.getDegree();

}

}

**this**.setPolynomialDegree(grad);

}

**public** **void** addMonom(Monom m) {

**if** (m.getCoefficient() == 0) {

**return**;

} **else** {

**if** (m.getDegree() > **this**.getPolynomialDegree()) {

**this**.polynomialDegree = m.getDegree();

**this**.polynom.add(0, m);

} **else** {

**boolean** ok = **false**;

**int** position = 0;

**for** (Monom a : **this**.getPolynom()) {

**if** (a.getDegree() == m.getDegree()) {

**this**.getPolynom().set(position, a.add(m));

ok = **true**;

**break**;

}

**if** (a.getDegree() < m.getDegree()) {

**break**;

}

position++;

}

**if** (!ok) {

**this**.getPolynom().add(position, m);

}

}

}

}

@Override

**public** String toString() {

String s = **new** String();

**int** firstCoef = (**int**) **this**.getFirstNotNull().getCoefficient();

**for** (Monom a : **this**.getPolynom()) {

s += a.toString();

}

**if** (firstCoef > 0)

**return** s.substring(1);

**else** {

**if** (s.equals(""))

**return** "0";

**else**

**return** s;

}

}

**public** Polinom addPolynomials(Polinom b) {

Polinom result = **new** Polinom();

**for** (Monom a : **this**.getPolynom()) {

result.addMonom(a);

}

**for** (Monom a : b.getPolynom()) {

result.addMonom(a);

}

result.updatePolynomial();

**return** result;

}

**public** Polinom subPolynomials(Polinom b) {

Polinom result = **new** Polinom();

**for** (Monom a : **this**.getPolynom()) {

result.addMonom(a);

}

**for** (Monom a : b.getPolynom()) {

result.addMonom(**new** Monom(a.getDegree(), -a.getCoefficient()));

}

result.updatePolynomial();

**return** result;

}

**public** **static** Polinom multiplyPolynomials(Polinom a, Polinom b) {

Polinom result = **new** Polinom();

**for** (Monom a1 : a.getPolynom()) {

**for** (Monom b1 : b.getPolynom()) {

Monom rez = a1.multiply(b1);

result.addMonom(rez);

}

}

result.updatePolynomial();

**return** result;

}

**public** Polinom dividePolynomials(Polinom b) **throws** IllegalArgumentException {

Polinom cat = **new** Polinom();

**if** (b.getPolynomialDegree() == 0) {

**if** (b.getFirstNotNull().getCoefficient() == 0) {

**throw** **new** IllegalArgumentException("Error: Divide by 0 is illegal");

} **else** {

**for** (Monom a : **this**.getPolynom()) {

cat.addMonom(**new** Monom(a.getDegree(),a.getCoefficient()/b.getFirstNotNull().getCoefficient()));

a.setCoefficient(0);

}

}

} **else** {

**while**(**this**.getPolynomialDegree() >= b.getPolynomialDegree()){

Polinom aux=**new** Polinom();

Monom d1=**this**.getFirstNotNull();

Monom i1=b.getFirstNotNull();

Monom firstDivide=d1.divide(i1);

cat.addMonom(firstDivide);

**for** (Monom imp : b.getPolynom()) {

aux.addMonom(firstDivide.multiply(imp));

}

**this**.setPolynom(**this**.subPolynomials(aux).getPolynom());

**this**.updatePolynomial();

}

}

**return** cat;

}

**public** Polinom derivePolynomial() {

Polinom result = **new** Polinom();

**for** (Monom a : **this**.getPolynom()) {

result.addMonom(a.derive());

}

**return** result;

}

**public** Polinom integratePolynomial() {

Polinom result = **new** Polinom();

**for** (Monom a : **this**.getPolynom()) {

result.addMonom(a.integrate());

}

**return** result;

}

**public** Monom getFirstNotNull() {

Monom result = **new** Monom();

**for** (Monom a : **this**.getPolynom()) {

**if** (a.getCoefficient() != 0) {

result = a;

**break**;

}

}

**return** result;

}

}

Clasa ***PolynomialUtils*** vine cu metode ajutătoare pentru interacționarea interfeței grafice cu sistemul de procesare al polinoamelor. Principala metodă a clasei este **parsePolynomial** care realizează transpunerea unui String într-un polinom definit ca în clasa Polinom. Celelalte metode sunt private și sunt **removeBlanks** (elimină spațiile dintr-un String), **divideString**(Împarte String-ul în mai multe substring-uri după semnul “+” care este în cazul de față delimitatorul între monoame) și metoda **matchPattern**(care verifică forma unui monom și este capabilă să identifice coeficentul monomului și gradul acestuia), și va returna un monom cu coeficientul și gradul identificate, după cazul de match. Codul pentru această clasă este următorul :

package Model;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import javax.swing.JOptionPane;

public class PolynomialUtils {

public Polinom parsePolynomial(String input) throws Exception {

System.out.println("Inputul este " + input);

Polinom result = new Polinom();

PolynomialUtils util = new PolynomialUtils();

input = input.replaceAll(" ", "");

String[] monomArray = util.divideString(input);

for (String monom : monomArray) {

Monom m;

try {

System.out.println(monom);

if (!monom.equals("")) {

if (monom.charAt(0) == '-') {

m = util.matchPattern(monom.substring(1));

m.setCoefficient(-m.getCoefficient());

} else {

m = util.matchPattern(monom);

}

System.out.println("Monomul este");

System.out.println(m.toString());

result.addMonom(m);

}

} catch (Exception e) {

System.out.println(e.getMessage());

JOptionPane.showMessageDialog(null, "The input is invalid. Provide valide data!","Error",JOptionPane.ERROR\_MESSAGE);

throw new Exception("");

}

}

System.out.println("Polinomul este:");

System.out.println(result.toString());

return result;

}

private String removeBlanks(String input) {

return input.replaceAll(" ", "");

}

private String[] divideString(String input) {

input = input.replaceAll("-", "+-");

String[] monomArray = input.split("(\\+)");

return monomArray;

}

private Monom matchPattern(String s) throws Exception {

String pattern[] = new String[7];

pattern[0] = "(^\\d+)(\\\*)(x)(\\^)(\\d+$)";

pattern[1] = "(^\\d+)(x)(\\^)(\\d+$)";

pattern[2] = "(^x)(\\^)(\\d+$)";

pattern[3] = "(^\\d+)(\\\*)(x$)";

pattern[4] = "(^\\d+)(x$)";

pattern[5] = "(^x$)";

pattern[6] = "(^\\d+$)";

int i = 0;

Pattern p = Pattern.compile(pattern[0]);

Matcher mtch = p.matcher(s);

for (i = 0; i < 7; i++) {

p = Pattern.compile(pattern[i]);

mtch = p.matcher(s);

if (mtch.find()) {

break;

}

}

Monom newMonom = new Monom();

if (i >= 7) {

throw new Exception("The input is invalid. Provide valide data!");

} else {

switch (i) {

case 0:

newMonom.setDegree(Integer.parseInt(mtch.group(5)));

newMonom.setCoefficient(Integer.parseInt(mtch.group(1)));

break;

case 1:

newMonom.setDegree(Integer.parseInt(mtch.group(4)));

newMonom.setCoefficient(Integer.parseInt(mtch.group(1)));

break;

case 2:

newMonom.setDegree(Integer.parseInt(mtch.group(3)));

newMonom.setCoefficient(1);

break;

case 3:

newMonom.setDegree(1);

newMonom.setCoefficient(Integer.parseInt(mtch.group(1)));

break;

case 4:

newMonom.setDegree(1);

newMonom.setCoefficient(Integer.parseInt(mtch.group(1)));

break;

case 5:

newMonom.setDegree(1);

newMonom.setCoefficient(1);

break;

case 6:

newMonom.setDegree(0);

newMonom.setCoefficient(Integer.parseInt(mtch.group(1)));

break;

}

}

return newMonom;

}

}

Clasa ***PolynomialModel*** este clasa reprezentativă din structura Model – View – Controller. Aceasta are 3 atribute, două String-uri pentru afișarea rezultatelor (rezultatul unei operații și restul în cazul împărțirii a două polinoame) și un String static final cu valoare “” pentru resetare și valori inițiale.

Principalele metode implementate sunt pentru operațiile matematice cu care operăm pe polinoame. Întrucât modelul metodei este identic pentru toate operațiile, vom prezenta metoda pentru suma a două polinoame:

**public** **void** add(String firstPolynomial, String secondPolynomial) {

PolynomialUtils util = **new** PolynomialUtils();

**try** {

Polinom p1 = util.parsePolynomial(firstPolynomial);

Polinom p2 = util.parsePolynomial(secondPolynomial);

Polinom result = p1.addPolynomials(p2);

m\_total = "" + result.toString();

} **catch** (Exception e) {

m\_total = "";

}

}

În cadrul metodei, vom apela metoda de parsePolynomial care transformă un String în polinom și apoi metoda pentru addPolynomials care realizează suma dintre polinoamele rezultate în urma parsării. String-ul pentru rezultat este pus în m\_total.

În pachetul **Controller,** avem o singură clasă, **PolynomialController** cu atribute un obiect de tip PolynomialView și un obiect PolynomialModel. Constructorul pentru o instanță a PolynomialController adaugă ascultători pentru butoanele din interfața grafică, în cadrul clasei definim și clase care implementează ActionListener și care execută în funcție de operația selectată, anumite procese de computație.

Codul pentru această clasă:

package Controller;

import java.awt.event.\*;

import javax.swing.JOptionPane;

import Model.PolynomialModel;

import View.PolynomialView;

public class PolynomialController {

private PolynomialModel m\_model;

private PolynomialView m\_view;

public PolynomialController(PolynomialModel model, PolynomialView view) {

m\_model = model;

m\_view = view;

view.addMultiplyListener(new MultiplyListener());

view.addDivideListener(new DivideListener());

view.addSumListener(new SumListener());

view.addSubListener(new SubListener());

view.addDeriveListener(new DeriveListener());

view.addIntegrateListener(new IntegrateListener());

view.addClearListener(new ClearListener());

}

class MultiplyListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

String firstPo = "";

String secondPo = "";

try {

firstPo = m\_view.getUserInputFirstPolynomial();

secondPo = m\_view.getUserInputSecondPolynomial();

m\_model.multiply(firstPo, secondPo);

m\_view.setResult(m\_model.getValue());

m\_view.setRestVisibility(false);

m\_view.setSecondPolynomialField(true);

} catch (Exception err) {

m\_view.showError(err.getMessage());

}

}

}

class DivideListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

String firstPo = "";

String secondPo = "";

try {

firstPo = m\_view.getUserInputFirstPolynomial();

secondPo = m\_view.getUserInputSecondPolynomial();

m\_model.divide(firstPo, secondPo);

m\_view.setRestVisibility(true);

m\_view.setResult(m\_model.getValue());

m\_view.setRest(m\_model.getRest());

m\_view.setSecondPolynomialField(true);

} catch (Exception err) {

m\_view.showError(err.getMessage());

}

}

}

class SumListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

String firstPo = "";

String secondPo = "";

try {

firstPo = m\_view.getUserInputFirstPolynomial();

secondPo = m\_view.getUserInputSecondPolynomial();

m\_model.add(firstPo, secondPo);

m\_view.setRestVisibility(false);

m\_view.setResult(m\_model.getValue());

m\_view.setSecondPolynomialField(true);

} catch (Exception err) {

m\_view.showError(err.getMessage());

}

}

}

class SubListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

String firstPo = "";

String secondPo = "";

try {

firstPo = m\_view.getUserInputFirstPolynomial();

secondPo = m\_view.getUserInputSecondPolynomial();

m\_model.sub(firstPo, secondPo);

m\_view.setRestVisibility(false);

m\_view.setResult(m\_model.getValue());

m\_view.setSecondPolynomialField(true);

} catch (Exception err) {

m\_view.showError(err.getMessage());

}

}

}

class DeriveListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

String firstPo = "";

// String secondPo="";

try {

firstPo = m\_view.getUserInputFirstPolynomial();

// secondPo=m\_view.getUserInputSecondPolynomial();

m\_model.derive(firstPo);

m\_view.setRestVisibility(false);

m\_view.setResult(m\_model.getValue());

m\_view.setSecondPolynomialField(false);

} catch (Exception err) {

m\_view.showError(err.getMessage());

}

}

}

class IntegrateListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

String firstPo = "";

// String secondPo="";

try {

firstPo = m\_view.getUserInputFirstPolynomial();

// secondPo=m\_view.getUserInputSecondPolynomial();

m\_model.integrate(firstPo);

m\_view.setRestVisibility(false);

m\_view.setResult(m\_model.getValue());

m\_view.setSecondPolynomialField(false);

} catch (Exception err) {

m\_view.showError(err.getMessage());

}

}

}

class ClearListener implements ActionListener {

public void actionPerformed(ActionEvent e) {

m\_view.reset();

m\_model.reset();

m\_view.setSecondPolynomialField(true);

}

}

}

În pachetul **View,** avem o clasă **PolynomialView,** care reprezintă implemetnarea interfeței grafice. Atributele pentru această clasa sunt:

**private** JTextField firstPolynomialtF = **new** JTextField();

**private** JTextField secondPolynomialtF = **new** JTextField();

**private** JTextField resulttF = **new** JTextField();

**private** JTextField resttF = **new** JTextField();

**private** JButton multiplyBtn = **new** JButton("Multiply");

**private** JButton divideBtn = **new** JButton("Divide");

**private** JButton addBtn = **new** JButton("Add");

**private** JButton subBtn = **new** JButton("Subtract");

**private** JButton deriveBtn = **new** JButton("Derive");

**private** JButton integrateBtn = **new** JButton("Integrate");

**private** JButton clearBtn = **new** JButton("Clear");

**private** JLabel restJL = **new** JLabel("Rest");

**private** PolynomialModel polModel;

Așadar, avem textfield-uri pentru introducerea celor două polinoame, pentru rezultat și eventual rest și butoane pentru operații. Label-ul pentru rest este un atribut al clasei, întrucât vizibilitatea acestuia este modificată în funcție de operația efectuatuă, astfel va fi vizibil doar dacă realizăm impărțirea polinoamelor.

Interfața grafică este construită in felul următor:

**public** PolynomialView(PolynomialModel model) {

**this**.setMinimumSize(**new** Dimension(700, 300));

polModel = model;

polModel.setValue(PolynomialModel.***INITIAL\_VALUE***);

resulttF.setText(polModel.getValue());

resulttF.setEditable(**false**);

resttF.setText(polModel.getRest());

resttF.setEditable(**false**);

JPanel content = **new** JPanel();

content.setLayout(**new** BoxLayout(content, BoxLayout.***PAGE\_AXIS***));

JPanel firstLabel = **new** JPanel();

firstLabel.add(**new** JLabel("Introduce the first polynomial"));

content.add(firstLabel);

content.add(firstPolynomialtF);

JPanel secondLabel = **new** JPanel();

secondLabel.add(**new** JLabel("Introduce the second polynomial"));

content.add(secondLabel);

content.add(secondPolynomialtF);

JPanel singleBtn = **new** JPanel();

singleBtn.add(clearBtn);

content.add(singleBtn);

JPanel buttons = **new** JPanel();

buttons.setLayout(**new** FlowLayout());

buttons.add(addBtn);

buttons.add(Box.*createHorizontalStrut*(30));

buttons.add(subBtn);

buttons.add(Box.*createHorizontalStrut*(30));

buttons.add(multiplyBtn);

buttons.add(Box.*createHorizontalStrut*(30));

buttons.add(divideBtn);

buttons.add(Box.*createHorizontalStrut*(30));

buttons.add(deriveBtn);

buttons.add(Box.*createHorizontalStrut*(30));

buttons.add(integrateBtn);

content.add(buttons);

content.add(**new** JLabel("Result"));

content.add(resulttF);

content.add(restJL);

content.add(resttF);

**this**.setContentPane(content);

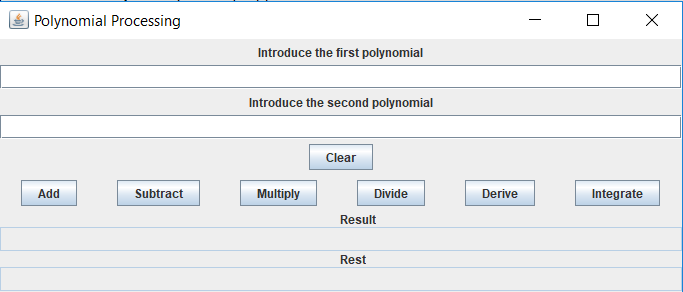
**this**.pack();

**this**.setTitle("Polynomial Processing");

**this**.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

}

În acest fel, interfața va arăta astfel:



Această clasă mai are metode pentru adăugarea de ascultători pentru butoane și pentru preluarea string-urilor din câmpurile pentru introducere de polinoame.

Exemplu:

**public** String getUserInputFirstPolynomial() {

**return** firstPolynomialtF.getText();

}

și

**public** **void** addMultiplyListener(ActionListener mal) {

multiplyBtn.addActionListener(mal);

}

În pachetul **PT2019.Tema1**  avem clasa **App**, de unde lansăm aplicația și **AppTest(**pentru testarea unitară) :

package PT2019.Tema1;

import Controller.PolynomialController;

import Model.PolynomialModel;

import View.PolynomialView;

public class App

{

public static void main( String[] args )

{

PolynomialModel model = new PolynomialModel();

PolynomialView view = new PolynomialView(model);

PolynomialController controller = new PolynomialController(model, view);

view.setVisible(true);

}

}

**Capitolul V. *Rezultate***

În acest capitol, vom prezenta scenariile de testare cu Junit pentru aplicația noastră.

1. *Adunare*

Vom lua primul polinom și al doilea polinom . Rezultatul adunării acestor polinoame este . În clasa AppTest vom avea următoarea:

**public** **void** testSum(){

Polinom p1 = **new** Polinom();

p1.addMonom(**new** Monom(3,2));

p1.addMonom(**new** Monom(2,4));

p1.addMonom(**new** Monom(1,5));

p1.addMonom(**new** Monom(0,3));

Polinom p2 = **new** Polinom();

p2.addMonom(**new** Monom(4,1));

p2.addMonom(**new** Monom(2,1));

p2.addMonom(**new** Monom(1,4));

Polinom p3 = **new** Polinom();

p3.addMonom(**new** Monom(4,1));

p3.addMonom(**new** Monom(3,2));

p3.addMonom(**new** Monom(2,5));

p3.addMonom(**new** Monom(1,9));

p3.addMonom(**new** Monom(0,3));

*assertTrue*(p3.equals(p1.addPolynomials(p2)));

}

1. *Scădere*

Vom lua primul polinom și al doilea polinom . Rezultatul scăderii acestor polinoame este . În clasa AppTest vom avea următoarea:

**public** **void** testSub(){

Polinom p1 = **new** Polinom();

p1.addMonom(**new** Monom(3,2));

p1.addMonom(**new** Monom(2,4));

p1.addMonom(**new** Monom(1,5));

p1.addMonom(**new** Monom(0,3));

Polinom p2 = **new** Polinom();

p2.addMonom(**new** Monom(4,1));

p2.addMonom(**new** Monom(2,1));

p2.addMonom(**new** Monom(1,4));

Polinom p3 = **new** Polinom();

p3.addMonom(**new** Monom(4,-1));

p3.addMonom(**new** Monom(3,2));

p3.addMonom(**new** Monom(2,3));

p3.addMonom(**new** Monom(1,1));

p3.addMonom(**new** Monom(0,3));

*assertTrue*(p3.equals(p1.subPolynomials(p2)));

}

1. Înmulțire

Vom lua primul polinom și al doilea polinom . Rezultatul înmulțirii acestor polinoame este . În clasa AppTest vom avea următoarea:

**public** **void** testMultiply(){

Polinom p1 = **new** Polinom();

p1.addMonom(**new** Monom(2,1));

p1.addMonom(**new** Monom(1,2));

p1.addMonom(**new** Monom(0,1));

Polinom p2 = **new** Polinom();

p2.addMonom(**new** Monom(1,1));

p2.addMonom(**new** Monom(0,2));

Polinom p3 = **new** Polinom();

p3.addMonom(**new** Monom(3,1));

p3.addMonom(**new** Monom(2,4));

p3.addMonom(**new** Monom(1,5));

p3.addMonom(**new** Monom(0,2));

*assertTrue*(p3.equals(Polinom.*multiplyPolynomials*(p1, p2)));

}

1. Împărțire

Vom lua primul polinom 2 și al doilea polinom . Rezultatul împărțirii acestor polinoame este câtul și restul 14. În clasa AppTest vom avea următoarea:

**public** **void** testDivide(){

Polinom p1 = **new** Polinom();

p1.addMonom(**new** Monom(2,2));

p1.addMonom(**new** Monom(1,3));

p1.addMonom(**new** Monom(0,5));

Polinom p2 = **new** Polinom();

p2.addMonom(**new** Monom(1,1));

p2.addMonom(**new** Monom(0,3));

Polinom cat=**new** Polinom();

cat.addMonom(**new** Monom(1,2));

cat.addMonom(**new** Monom(0,-3));

Polinom rest=**new** Polinom();

rest.addMonom(**new** Monom(0,14));

Polinom rezCat = p1.dividePolynomials(p2);

*assertTrue*(p1.equals(rest));

*assertTrue*(rezCat.equals(cat));

}

1. Derivare

Vom lua polinomul . Rezultatul derivării acestui polinom este . În clasa AppTest vom avea următoarea:

**public** **void** testDerive(){

Polinom p1 = **new** Polinom();

p1.addMonom(**new** Monom(4,3));

p1.addMonom(**new** Monom(3,8));

p1.addMonom(**new** Monom(2,5));

p1.addMonom(**new** Monom(1,4));

p1.addMonom(**new** Monom(0,3));

Polinom p2 = **new** Polinom();

p2.addMonom(**new** Monom(3,12));

p2.addMonom(**new** Monom(2,24));

p2.addMonom(**new** Monom(1,10));

p2.addMonom(**new** Monom(0,4));

*assertTrue*(p2.equals(p1.derivePolynomial()));

}

1. Integrare

Vom lua polinomul . Rezultatul derivării acestui polinom este . În clasa AppTest vom avea următoarea:

**public** **void** testIntegrate(){

Polinom p1 = **new** Polinom();

p1.addMonom(**new** Monom(4,3));

p1.addMonom(**new** Monom(3,8));

p1.addMonom(**new** Monom(2,5));

p1.addMonom(**new** Monom(1,4));

p1.addMonom(**new** Monom(0,3));

Polinom p2 = **new** Polinom();

p2.addMonom(**new** Monom(5,3/5.0));

p2.addMonom(**new** Monom(4,2));

p2.addMonom(**new** Monom(3,5/3.0));

p2.addMonom(**new** Monom(2,2));

p2.addMonom(**new** Monom(1,3));

*assertTrue*(p2.equals(p1.integratePolynomial()));

}

1. Testare pentru parsare

Vom introduce un string și vom verifica dacă polinomul parsat este cel correct:

**public** **void** testParsing(){

String text = **new** String("-x^2 + x +3\*x^2+++2x +-5");

Polinom p1 = **new** Polinom();

p1.addMonom(**new** Monom(2,2));

p1.addMonom(**new** Monom(1,3));

p1.addMonom(**new** Monom(0,-5));

PolynomialUtils util = **new** PolynomialUtils();

Polinom rez;

**try**{

rez = util.parsePolynomial(text);

*assertTrue*(p1.equals(rez));

}

**catch**(Exception e){

System.***out***.println(e.getMessage());

*assertTrue*(**true**);

}

}

**Capitolul VI. *Concluzii***

Tema prezentată impune cunoașterea și aplicarea noțiunilor din paradigma programării orientate pe obiect. Așadar, rezolvarea cerinței permite aprofundarea și dezvoltarea capacităților de a scrie cod Java, identificarea cauzelor și tratarea problemelor ce pot apărea în procesul de implementare. Totodată, datorită implementării și a unei interfețe grafice, programatorul trebuie să conceapă o interfață inteligibilă prin care utilizatorul poate experimenta toate funcționalitățile puse în evidență.

Aplicația concepută este utilă utilizatorilor care au nevoie să aplice operații simple matematice asupra polinoamelor. Această aplicație poate avea și dezvoltări ulterioare, cum ar fi introducerea de diferite funcționalități noi, cum ar fi: găsirea rădăcinilor unui polinom, trasarea graficului funcției polinomiale, determinarea relațiilor lui Viette pentru polinoame de ordinul n etc.

**Capitolul VII. *Bibliografie***

[1] [www.matematicon.ro](http://www.matematicon.ro)

[2] [www.wikipedia.org](http://www.wikipedia.org)

[3] [www.tutorialspoint.com](http://www.tutorialspoint.com)