DOCUMENTATIE

TEMA 1

*Calculator de polinoame*

*Nume student: SZAKACS EMMA-EVELIN*

*Grupa: 30221*

**CUPRINS**

[**1.** **Obiectivul temei** 3](#_Toc97489120)

[**2.** **Analiza problemei, modelare, scenarii, cazuri de utilizare** 3](#_Toc97489121)

[**3.** **Proiectare** 5](#_Toc97489122)

[1. Diagrama UML 5](#_Toc97489123)

[3.2 Structuri de date folosite 6](#_Toc97489124)

[**4.** **Implementare** 7](#_Toc97489125)

[4.1.Clase si metode 7](#_Toc97489126)

[4.2. GUI 11](#_Toc97489127)

[**5.** **Rezultate** 12](#_Toc97489128)

[**6.** **Concluzii** 14](#_Toc97489129)

[**7.** **Bibliografie** 14](#_Toc97489130)

# **Obiectivul temei**

**Obiectivul principal** al acestei teme reprezinta proiectarea si implementarea unui sistem de procesarea a polinoamelor cu coeficienti reali(de tip double) si exponenti intregi.

Obiectivul acestei teme a fost de a proiecta un calculator de polinoame de coeficienti de reali(de tip double). Calculatorul trebuie sa poata efectua diferite operatii intre doua polinoame, cum ar fi adunarea, scaderea, inmultirea si impartirea. De asemenea operatiile de derivare si integrare se pot efectua asupra primului polinom.

**Obiectivele secundare** care contribuie la realizarea obiectivului principal sunt:

|  |  |
| --- | --- |
| 1. Analiza si intelegerea problemei. | Capitolul 1 |
| 2. Folosirea conceptelor OOP, cum ar fi incapsularea si definirea unor clase potrivite pentru descompunrea problemei in subprobleme. | Capitolul 3 |
| 3. Implementarea operatiilor de adunare, scadere, inmultire si impartire dintre doua polinoame cu coeficienti reali(de tip double). | Capitolul 4 – Implementare |
| 1. Implementarea operatiilor de integrare si derivare asupra primului polinom | Capitolul 4 – Implementare |
| 1. Folosirea unei interfete grafice “User Friendly” – folosind Java Swing | Capitolul 4 -Implementare Capitolul 5 - Rezultate |
| 1. Testarea unitara (folosind JUnit) pentru verificarea rezultatelor. | Capitolul 5 - Rezultate |
|  |  |

# [**Analiza problemei, modelare, scenarii, cazuri de utilizare**](file:///C:\Users\Emma\Downloads\PT2021-2022_Documentation_Template.doc#_Toc95297886)

In matematica, un polinom este o expresie construita dintr-una sau mai multe variabile si constante. Polinoamele sunt construite din termini numiti monoame, care sunt alcatuite dintr-o constanat(numita coefficient) inmultita cu o variabila care care poate avea sau nu un exponent. Exponentul enei variabile dintr-un monom este egal cu gradul acelei variabile in acel monom.

Toate polinoamele de o variabila sunt echivalente cu un polinom de forma:

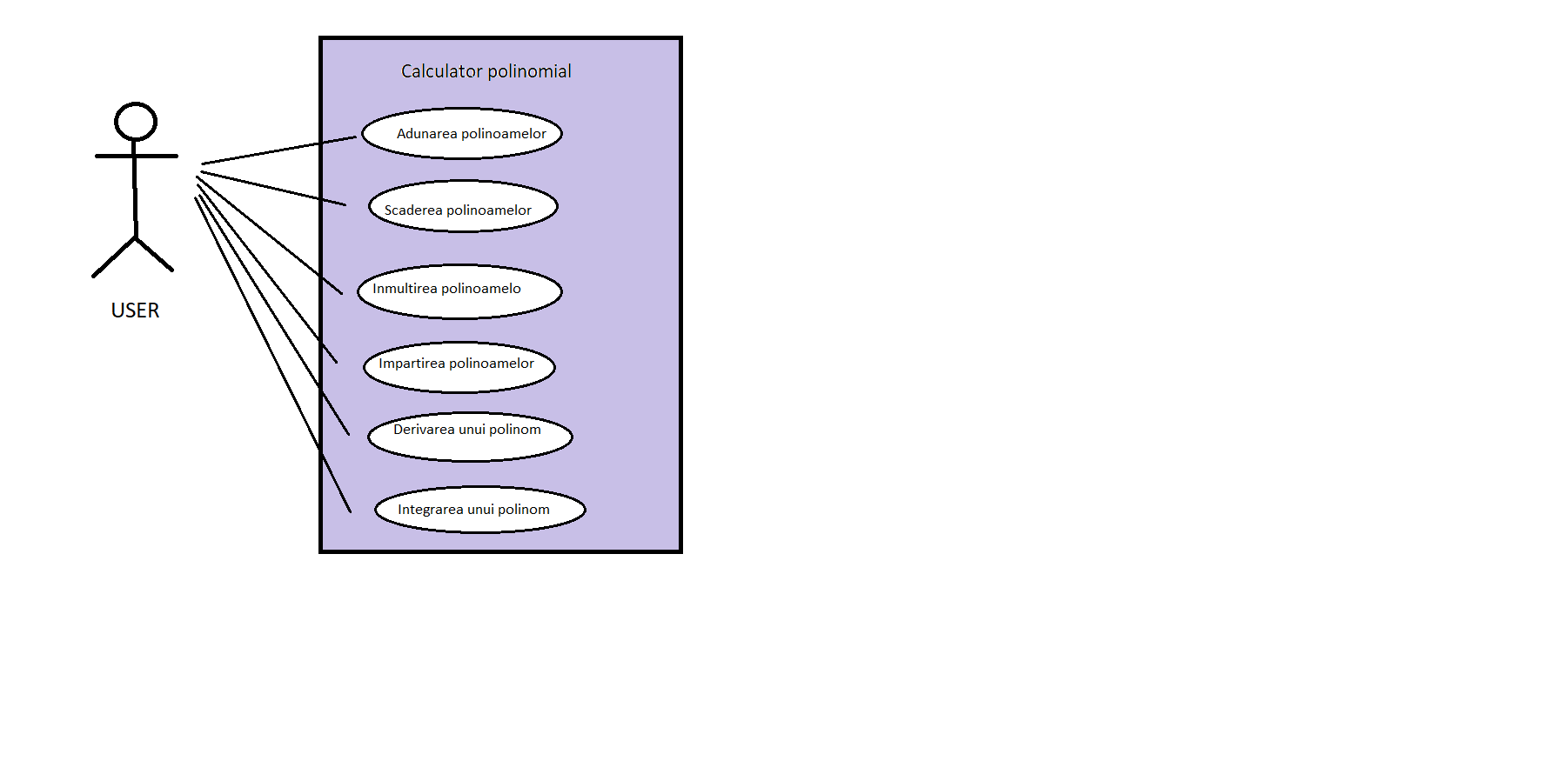
P = an\*X^n + …. + a2\*X^2 + a1\*X^1

Exista o multitudine de metode prin care se poate realiza calculatorul. Introducerea datelor, utilizarea datelor, aspectul interfetei si alte aspecte legate de implementare tin de alegerile fiecarui programator.

Pentru separarea unui string intr-o forma convenabila se poate folosi un Regex, aceasta fiind o metoda mai grea, dar mai eficienta si mai scurta. De asemenea regexul este util pentru identificarea inputului gresit introdus de catre utilizator. In cazul in care se introduce un input care nu respecta forma unui polininom, se va genera un mesaj de eroare.

**Cazuri de utilizare**

Actorul principal in acest scenario este utilizatorul. Utilizatorul trebuie sa poata introduce doua polinoame in interfata grafica. Inputul poate fi scris de la tastatura sau din butoanele care apar in interfata. Pentru realizarea uneia din operatiile “Add”, “Subtract”,”Multiplicate”,”Divide”,”Integrate” sau “Derivative” se va apasa unul din butoanele care poarta numele operatiei care doreste sa fie executata. Dupa apasarea butonului se efectueaza calculele si se afiseaza rezultatul operatiei.



**Modelarea proiectului**

Pentru modelarea proiectului am folosit design-ul MVC( Model View Controller). Astel am creeat 3 obiecte diferite pentru model, view si controller.

*Modelul* contine datele aplicatiei, nici de cum logica care descrie cum vor fi prezentate datele utilizatorului.

*View-u*l prezinta prezinta utilizatorului datele modelului. View-ul stie sa acceseze datele modelului, dar nu stie ce semnifica ele sau ce poate face utilizatorul ca sa le modifice.

*Controller-ul* este legatura dintre view si model. Asculta evenimentele declansate de view sau alte surse si reactioneaza la respectivele evenimente.

Diagram

Description automatically generated

# **Proiectare**

## Diagrama UML

Unified Modeling Language sau UML pe scurt este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora ( numite și obiecte ). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT.

Diagrama UML generate pentru proiectul acesta este urmatoarea:

Graphical user interface

Description automatically generated

## **Structuri de date folosite**

Listele in Java ofera mentinerea colectiei ordonate. Listele contin metoda de inserare, de actualizare,de stergere si de cautare a elementelor, fiind mult mai eficiente decat array-urile. De asemenea listele permit elemente duplicate si pot retine elemente nule.

In acest proiect am folosit o lista pentru retinerea monoamelor unui polinom, implementata cu ArrayList. De asemenea, la operatia de impartire a polinoamelor a fost mult mai usor sa returnez o lista care contine catul si restul.

# **Implementare**

## 4.1.Clase si metode

1. **Clasa Controller** contine logica pentru actualizarea modelului si/sau view-ului la introducerea datelor de catre utilizator. Acesta are un constructor care primeste modelul si view-ul, iar in functie de ce buton apasa utilizatorul apeleaza metoda *actions* care preia inputul de la utilizator si apeleaza metoda *getResult* din model pentru a genera rezultatul. In cazul in care string-ul este gol, in textFieldResult se va pune “0”
2. **Clasa Model** contine metoda getResult care primeste trei siruri de caractere: inputul de la primul textField, inputul de la al doilea textField si numele operatiei. Cele doua inputuri se convertesc la tipul Polynomial prin apelul la functia *stringToPolynomial* care contine apelul *la Regex.* In functie de operatia care se trimite ca string se genereaza rezultatul. Deoarece in cazul impartirii, rezultatul impartirii este o lista de polinoame, mai exact catul si restul, a trebuit sa iau cazul respective separat. Asadar, in cazul in care opeartia este “Divide” voi afisa catul apoi voi scrie restul(tot in acelasi textFieldResult).
3. **Clasa Monomial** contine ca variabile instanta exponentul(de tip intreg) si coeficientul(de tip flotant) ale unui termen dintr-un polinom. De asemenea in aceasta clasa se metoda toString pe care am folosit-o la afisare.
4. **Clasa Polynomial** are ca variabila instanta o lista de monoame si getter si setter pentru aceasta. De asemenea, am implementat doua metode s*ortMonomial* si *sortExponentsDesc* pentru sortarea polinoamelor pe care le am utilizat ulterior pentru efectuarea operatiilor.
5. **Clasele SortByExpDesc si SortByExp** implementeaza interfata Comparator si sunt folosite la sortarea crescatoare, respective descrescatoare a polinoamelor in functie de exponent.
6. **Clasa Operations** contine metodele pentru realizarea operatiilor de adunare, scadere, inmultire, impartire, integrare si derivare.

Metoda Add primeste 2 polinoame si le aduna. Cu doua for-uri parcurg cele doua polinoame, iar cand gasesc polinoame cu exponent egal, le adun. In cazul in care unul din polinoame a ajuns la final, iar celalalt nu, adun la rezultat termenii ramasi. La final apelez metoda deleteZero ca sa sterg elementele care au coeficientul zero.

Table, calendar

Description automatically generated

**Metoda Subtract** se bazeaza pe acelasi principiu, ca metoda Add, doar ca se va face scaderea celor doua polinoame.

Table

Description automatically generated

**Metoda Multiply** primeste doua polinoame si le inmulteste. Dupa inmultire vor rezultata termini care au coeficientii egali. In cazul acesta am folosit doua for-uri ca sa parcurg sirul iar termenii cu acelasi exponent ii adaug la primul termen care are exponentul respectiv, iar la restul le dau remove.

Table

Description automatically generated

**Metoda Derivate** primeste un polinom si il deriveaza. Metoda este una destul de simpla. Parcurg polinomul monom cu monom derivand termenii.

Graphical user interface, table

Description automatically generated

**Metoda Integrate** merge pe acelasi principiu ca metoda Derivate, singura diferenta fiind faptul ca fiecare monom va fi integrat.

Graphical user interface, application, table

Description automatically generated

Metoda Division este cea mai complexa dintre toate. Metoda primeste ca parametrii doua polinoame si returneaza o lista de polinoame care contine catul si restul impartirii celor doua polinoame. Am urmat pasii din prezentarea temei pentru a realiza operatia. Mai intai am sortat polinoamele prin apelul la metoda sortExponentsDesc. Am considerat ca polinomul introdus in primul textField este polinomul care urmeaza sa fie impartit, iar polinomul introdus in al doilea textField va fi impartitorul. In cazul in care al doilea polinom are cel mai mare grad mai mare decat cel mai mare grad al primului polinom catul va fi zero, iar restul va fi al doilea polinom. Altfel, am considerat restul egal cu primul polinom. Intr-o bucla while am efectuat calculele matematice pana cand gradul restului ajunge sa fie mai mic decat gradul polinomului al doilea. Calculele matematice au presupus apelul la metoda de inmultire, adunare si scadere. Ceea ce mi-a pus dificultati a fost sa imi dau seama ca inainte de toate calculele trebuie sa sortez descrescator dupa exponent polinomul reminder pentru ca toate calculele sa iasa correct.

Table

Description automatically generated

1. **Clasa Regex** contine metoda result care primeste un string si returneaza un polinom. Acesta metoda parseaza stringul astfel incat sa se obtina un polinom. Am folosit regex-ul ca sa salvez fiecare subexpresie intr-un grup. Primul pattern separa polinomul in mai multe monoame, iar al doilea separa fiecare monom in coefficient, variabila si exponent(reprezentand grupuri).

## 4.2. GUI

Intergafa grafica am implementat-o in clasa View care extinde clasa Frame. Inainte de toate am dat ca variabile instanta toate butoanele care vor aparea in interfata, textField-urile, JFrame-ul, label-urile si panel-urile. In partea de constructor apelez toate metodele pentru implementarea interfetei. Am impartit interfata in patru panel-uri. Am folosit ca si layout un BorderLayout. La inceput am pus panel-ul pentru input si rezultat, in centru am pus panel-ul cu butoanele pentru cifre, in partea de vest am pus panel-ul pentru butoanele de semne, iar in partea de jos am pus butoanele pentru operatii.

Pentru scrierea efectiva a plinoamelor asupra carora urmeaza sa se efectueze operatii se poate scrie prin folosirea butoanelor din interfata, dar si prin scrierea de la standard input(tastatura).

Am implementat o metoda chooseTextField pentru ca utilizatorul sa poata alege in ce textField sa scrie. Pentru fiecare textField am adaugat un focusListener. Cand utilizatorul apasa pe un textField se va pune focus-ul pe resprectivul textField, se va da remove la toate action listener-urile butoanelor prin metoda removeActionListeners( in cazul in care au activate ActionListeners), iar apoi se vor activa pentru textField-ul respectiv.

O metoda suplimentara pe care am implementat-o pentru imbunatatirea aspectului interfetei este mouseMove. Metoda apeleaza o alta metoda mouseMoveOneButton pentru fiecare buton. mouseMoveOneButton adauga un MouseListener butonului respective. Cand mouse-ul este pe buton, I se schimba culoarea butonului, iar cand mouse-ul este mutat, culoarea butonului revine la culoarea initiala.

Table

Description automatically generated Graphical user interface, table

Description automatically generated

In plus, in clasa View am pus settere pentru butoanele de operatii pe care o sa le folosesc apoi in model pentru obtinerea rezultatului in functie de butonul apasat de catre utilizator.

Pentru fiecare button, s-a făcut o metodă nouă care implică ActionEvent. Astfel, de fiecare dată când are loc o acțiune la un buton, de exemplu a fost apăsat, ActionEvent-ul denumit e, transmite informația la ActionListener care așteaptă astfel de informații. Apoi, au loc evenimentele ce se afla în metodă respectiva

# **Rezultate**

Pentru testarea codului am folosit JUnit care este un framework pentru testarea limbajului de programare Java. Am creeat o clasa noua OperationsTest care contine teste pentru toate operatiile. Clasa contine ca variabile instanta modelul, nrTesteEcecutate, nrTest(numarul testului curent) si nrTesteCuSucces. Inaine de toate(@BeforeAll) am dat modelului un nou model ca sa am access la taoate metodele si variabilele din model. La final (@AfterAll) voi afisa numarul testelor executate si numarul testelor executate cu success. Toate metodele pentru testarea operatiilor arata in felul urmator:

Graphical user interface, text, application

Description automatically generated

Cu diferenta ca in functie de operatie se va schimba stringul “Add” in numele operatiei specifice testului.

Metoda ProvideInput va fi diferita la pentru fiecare test, dar va fi dupa modelul urmator:

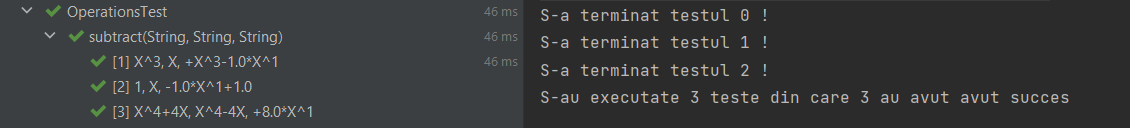
Text

Description automatically generated  
Rezultatul testului pentru adunare pentru exemplele date:

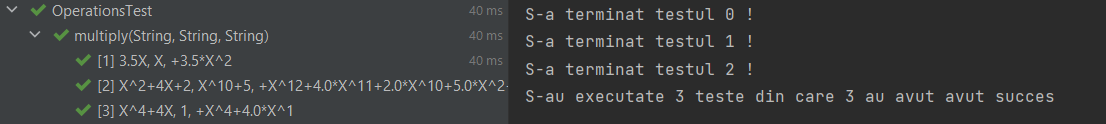
Graphical user interface

Description automatically generated

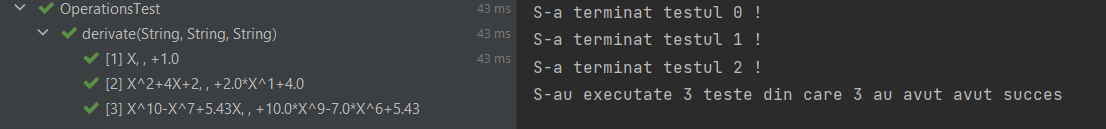
Rezultatul testului pentru scadere pentru exemplele date:

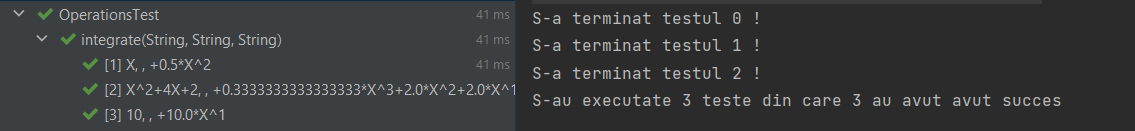


Rezultatul testului pentru inmultire pentru exemplele date:

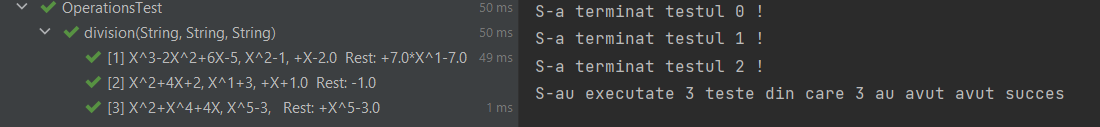


Rezultatul testului pentru derivare pentru exemplele date:



Rezultatul testului pentru integrare pentru exemplele date: 

Rezultatul testului pentru impartire pentru exempele date:



# **Concluzii**

Prin urmare, acest proiect a fost util pentru aprofundarea cunostintelor acumulate la limbajul de programare Java. Am invatat cum sa aranjez interfata grafica astfel incat sa arate si bine, nu doar sa mearga. De asemenea, mi-am reamntit cum sa folosesc testarea unitara cu Junit.

Ca dezvoltare ulterioara s-ar mai putea lucra la interfata grafica. O idee ar fi implementarea interfetei grafice in asa fel incat utilizatorul sa aleaga mai intai opeartia pe care vrea sa o efectueze, iar in functie de raspunsul lui sa apara un textField pentru derivare si integrare sau doua textField-uri pentru adunare, scadere, inmultire si impartire. De asemenea, in cazul impartirii ar fi mai ok daca ar fi doua textField-uri: unul pentru cat, iar celalalt pentru rest.

Ceea ce s-ar mai putea imbunatati, folosind regex-ul, este atunci cand utilizatorul introduce ceva gresit, sa se genereze un mesaj “Input gresit” si sa se stearga automat respectivul input, cerandu-i sa il introduca din nou corect.

# **Bibliografie**

* Java Swing: <https://docs.oracle.com/javase/tutorial/uiswing/>
* Junit: <https://www.vogella.com/tutorials/JUnit/article.html>
* Cursul de OOP din semestrul 1
* Regex: <https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html>
* MVC: <https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm>