**Documentație tema 1**

**Sistem de prelucrare polinoame**

Nume și prenume: Corbuleac Costel

Grupa: 30227

Profesor laborator: PhD Pop Cristina Bianca

Conținut

[1. Cerințe funcționale 4](#_Toc4134753)

[2. Obiective 4](#_Toc4134754)

[2.1. Obiectivul principal 4](#_Toc4134755)

[2.2. Obiective secundare 4](#_Toc4134756)

[2.2.1. Dezvoltarea cazurilor de utilizare și a scenariilor 4](#_Toc4134757)

[2.2.2. Alegerea structurilor de date 4](#_Toc4134758)

[2.2.3. Împărțirea pe clase 5](#_Toc4134759)

[2.2.4. Dezvoltarea algoritmilor 5](#_Toc4134760)

[2.2.5. Implementarea soluției 5](#_Toc4134761)

[2.2.6. Testare cu JUnit 5](#_Toc4134762)

[3. Analiza problemei 5](#_Toc4134763)

[3.1. Diagrama use-case 5](#_Toc4134764)

[3.2. Operația de adunare 6](#_Toc4134765)

[3.3. Operațiile de scădere și înmulțire 6](#_Toc4134766)

[3.4. Operația de împărțire 6](#_Toc4134767)

[3.5. Operația de derivare 6](#_Toc4134768)

[3.6. Operația de integrare 7](#_Toc4134769)

[3.7. Considerente generale 7](#_Toc4134770)

[4. Proiectare 7](#_Toc4134771)

[4.1. Structuri de date 7](#_Toc4134772)

[4.2. Diagrama de clase 8](#_Toc4134773)

[4.3. Diagrama de pachete 9](#_Toc4134774)

[4.4. Algoritmi 9](#_Toc4134775)

[4.4.1. Metoda de parsing createPolinom(String polinom) 9](#_Toc4134776)

[4.4.2. Metoda addition 9](#_Toc4134777)

[4.4.3. Metoda substraction 10](#_Toc4134778)

[4.4.4. Metoda multiplication 10](#_Toc4134779)

[4.4.5. Metoda division 10](#_Toc4134780)

[4.4.6. Metoda derivate 10](#_Toc4134781)

[4.4.7. Metoda integrate 11](#_Toc4134782)

[5. Implementare 11](#_Toc4134783)

[6. Testare 11](#_Toc4134784)

[7. Concluzii 12](#_Toc4134785)

[8. Bibliografie 12](#_Toc4134786)

# Cerințe funcționale

**Textul problemei :**

* Propuneţi, proiectaţi şi implementaţi un sistem de procesare a polinoamelor de o singură variabilă cu coeficienţi întregi.

**Modalitate de implementare :**

* Se vor citi unul sau două polinoame ca şiruri de caractere (String), iar apoi se vor procesa şi reprezenta sub forma unei liste de monoame pentru a putea realiza operațiile corespunzătoare. Datele de intrare, în acest caz cele două polinoame, trebuie introduse după următorul tipar “anx^n + an-1x^(n-1) + … + a1x^1 + a0x^0”, unde n aparține mulțimii numerelor naturale, an,, sau coeficientul, poate lua orice valoare din mulțimea numerelor întregi, iar x este singura variabilă a polinomului. String-urile vor fi introduse prin cadrul unui GUI (Graphical User Interface), prin două câmpuri text care vor avea un text temporar menit spre a ghida utilizator atunci când acesta dorește să introducă datele. La apăsarea oricărui buton pentru operațiile implementate (Adunare, Scădere, Înmulțire, Împărțire, Derivare, Integrare) se va extrage textul din căsuțele text și se vor crea cele două polinoame ca set de monoame.

# Obiective

## Obiectivul principal

* Principalul obiectiv este respectarea cerinței de lucru, și anume de a proiecta și implementa sistemul de prelucrare a două polinoame introduse de utilizator prin mediul grafic și returnarea rezultatelor în aceeași interfață utilizator în funcție de operația aleasă de cel ce introduce datele.

## Obiective secundare

### Dezvoltarea cazurilor de utilizare și a scenariilor

* Stabilirea modului în care datele pot fi introduse de către utilizator și acoperirea tuturor posibilităților de introducere a polinoamelor. În același timp utilizatorul trebuie informat dacă datele introduse de acesta nu sunt valide, sau dacă nu au fost introduse.

### Alegerea structurilor de date

* Pentru a putea reprezenta un polinom într-un sistem informatic se vor abstractiza datele și apoi se vor implementa cele 6 operații cu ajutorul noului model de date realizat.

### Împărțirea pe clase

* Aplicația va fi împărțită pe clase după modelul MVC ( Model – View – Controller ).

### Dezvoltarea algoritmilor

* Algoritmii vor fi dezvoltați astfel încât să reproducă operațiile matematice pe polinoame.

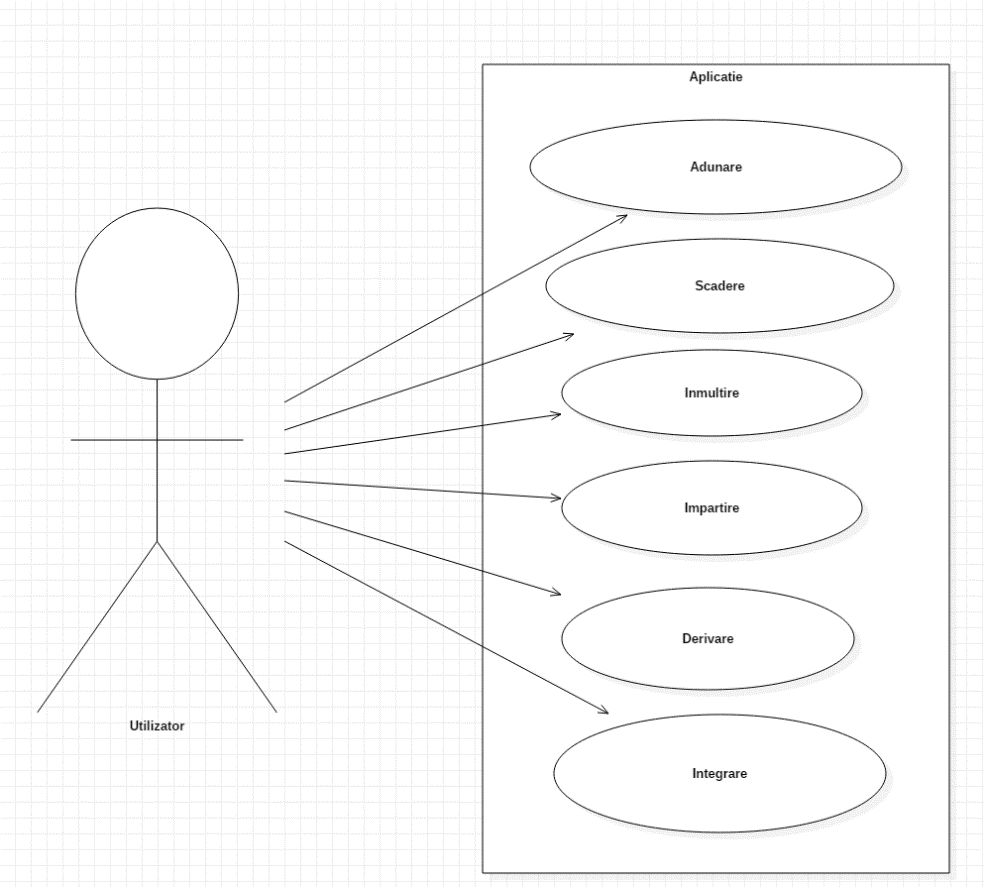
### Implementarea soluției

### Testare cu JUnit

* Se va crea un pachet pentru testarea operațiilor pe polinoame prin exemple gata calculate.

# Analiza problemei

## Diagrama use-case



## Operația de adunare

* Utilizatorul dorește să efectueze operația de adunare aplicată asupra celor două polinoame.
* Polinomul numărul #1 va fi introdus în căsuța etichetată cu ” Polynomial #1” și în care se află textul temporar "Introduce the first polynomial...", iar polinomul numărul #2 va fi introdus în căsuța etichetată cu ” Polynomial #2” și în care se află textul temporar "Introduce the second polynomial...".
* Se va alege operația “Addition” din câmpul cu cele șase butoane.
* Dacă datele de intrare sunt valide, atunci se va afișa rezultatul adunării în căsuța etichetată “Operation”. Altfel va apărea o căsuța de eroare ce va indica utilizatorului care este primul polinom invalid și ce reguli violează acesta, utilizatorul având posibilitatea să modifice erorile inițiale.

## Operațiile de scădere și înmulțire

* Procesul prin care se execută aceste două operații este asemănător cu cel de la operația de adunare, singura diferență fiind selectarea butoanelor. În cazul scăderii se alege butonul “Subtraction”, iar în cazul înmulțirii se va alege butonul inscripționat “Multiplication”.

## Operația de împărțire

* Și această operație se desfășoară asemănător cu operația de adunare. Se alege butonul “Division” și apoi câtul operației va fi afișat în TextField-ul etichetat “Operation”. Pe lângă această căsuță text și această etichetă vor mai apărea o etichetă denumită “Remainder” și o altă căsuță text în care se va afișa restul împărțirii. În același timp, în cadrul acestei operații, mai apare o condiție pentru polinomul #2 care obligă utilizatorul să introducă un polinom diferit de 0 pentru a se putea realiza cu succes împărțirea. Căsuța “Remainder” dispare atunci când utilizatorul alege să efectueze altă operație.

## Operația de derivare

* Utilizatorul dorește să efectueze operația de adunare aplicată asupra unui polinom.
* Polinomul numărul #1 va fi introdus în căsuța etichetată cu ” Polynomial #1” și în care se află textul temporar "Introduce the first polynomial...”.
* Se va alege operația “Derivation” din câmpul cu cele șase butoane.
* Dacă datele de intrare sunt valide, atunci se va afișa rezultatul derivării în căsuța etichetată “Operation”. Altfel va apărea o căsuța de eroare ce va indica ce reguli violează polinomul, utilizatorul având posibilitatea să modifice erorile inițiale.
* În căsuța etichetată cu ” Polynomial #2” și în care se află textul temporar "Introduce the second polynomial..." nu este obligatorie introducerea unei valori datorită faptului că operația de derivare se aplică pe primul polinom. De asemenea, dacă utilizatorul introduce un polinom în această căsuță text, nu se va produce un mesaj de eroare, dar polinomul va fi ignorat de GUI.

## Operația de integrare

* Asemănătoare cu operația de derivare. Pentru a se afișa rezultatul acestei operații se va alege butonul “Integration”.

## Considerente generale

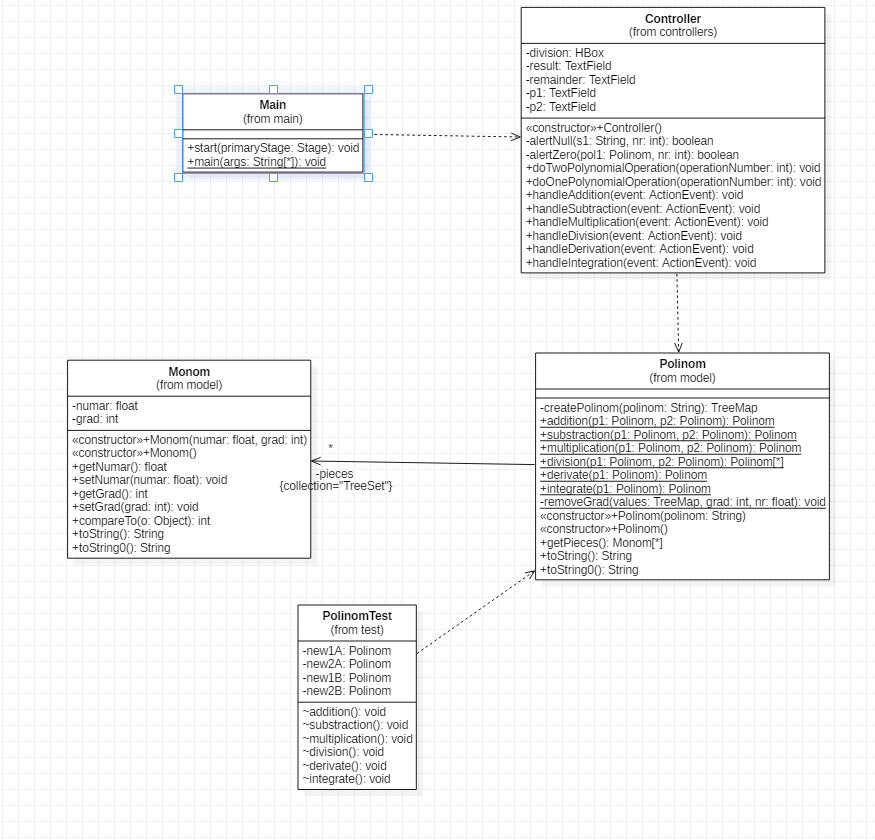
* Deși programul este realizat pentru polinoame cu coeficienți întregi, la operațiile de împărțire și integrare vor rezulta de multe ori valori reale, pe care le-am reprezentat sub forma unor variabile cu virgulă flotantă. Monoamele ce au coeficienți numere reale vor fi afișate ca și numere cu două zecimale. În acest mod se asigură atât păstrarea aspectului interfeței grafice cât și corectitudinea operațiilor.

# Proiectare

## Structuri de date

* Pentru parsarea polinoamelor și împărțirea inițială în grad și coeficient am decis să folosesc un TreeMap cu gradul ca și cheie și coeficientul ca valorii, datorită faptului că fiecare polinom are un singur coeficient pentru un anumit grad. Pentru stocarea monoamelor din cadrul unui polinom am ales să folosesc un TreeSet sortat descrescător în ordinea gradului. Această proprietate facilitează atât înmulțirea cât și împărțirea a două polinoame. Am ales un set din cauza unicității după grad a fiecărui monom. Un ArrayList este folosit în cadrul operației de împărțire pentru a stoca și returna două polinoame reprezentând câtul și restul operației. Motivul alegerii acestei structuri este acela că un ArrayList conține mai multe metode folositoare și este mai bine implementat decât un simplu vector.

## Diagrama de clase



## Diagrama de pachete

resources/fxml

main

test

model

controllers

## Algoritmi

### Metoda de parsing createPolinom(String polinom)

* Această metodă preia un String și înlocuiește fiecare “-” cu “+-“ apoi împarte polinomul după caracterul “+” . Astfel va rezulta un tablou unidimensional de monoame prin care iterez cu un foreach. Apoi pentru fiecare string monom se modifică string-ul dacă acesta conține cazuri speciale de monoame. Dacă monomul este de forma “-x^n” se înlocuiește cu un monom “-1x^n”, dacă monomul este de forma “x^n” se înlocuiește cu un monom “1x^n”, dacă monomul este de forma “a1x” se înlocuiește cu un monom “a1x^1” și dacă monomul este de forma “a0” se înlocuiește cu un monom “a0x^0”. Acest operații se realizează pentru a aduce orice monom la forma “anx^n”. În continuare se realizează un al doilea split după “x^”. După acest split, dacă polinomul este unul valid vom avea două string-uri, primul ce va conține coeficientul și al doilea ce va conține gradul. Dacă prima componentă este un număr flotant și a doua un număr natural, atunci se va realiza parsarea cu succes în variabilele locale “nr” și “grad”. În continuare se verifică dacă există deja în TreeMap gradul introdus și adună coeficientul actual la cel existent. În cazul în care această sumă de coeficienți are ca rezultat 0, se șterge din TreeMap gradul respectiv pentru a nu avea coeficienți de 0.
* În urma celor spuse reiese că în cazul în care gradul monomului este 1, monomul se poate introduce și sub forma “a1x”, iar atunci când monomul are gradul 0, acesta se poate reprezenta ca și ”a0”.

### Metoda addition

* Această metodă realizează adunare dintre două polinoame și se folosește de afișarea unui polinom, care are același format cu string-ul de citire al unui polinom. Datorită aceste proprietăți pot concatena string-urile de afișare ale celor două polinoame și să îl ofer ca parametru constructorului, care apelează metoda de parsare prezentată anterior. Astfel că adunarea este realizată în mod direct de metoda de parsare.

### Metoda substraction

* Această metodă realizează scăderea dintre două polinoame. Se înmulțește coeficientul fiecărui monom din cel de-al doilea polinom cu -1 și se apelează metoda de adunare.

### Metoda multiplication

* Am parcurs ambele TreeSet-uri de monoame cu câte un foreach, am înmulțit coeficienții, am adunat gradele și am adăugat tupla grad, coeficient într-un TreeMap la care verific dacă există deja gradul, pentru a aduna coeficienții asemănător cu algoritmul de parsare.

### Metoda division

*public static* ArrayList<Polinom> division(Polinom p1, Polinom p2) {  
 ArrayList<Polinom> list = *new* ArrayList<>();  
 Polinom remainder = *new* Polinom(p1.toString());  
 Polinom result = *new* Polinom("0");  
 Polinom polinomToSubstract;  
 *int* difference = remainder.getPieces().first().getGrad() - p2.getPieces().first().getGrad();  
 *float* division;  
 *while* (remainder.getPieces().size() > 0 && difference >= 0) {  
 division = remainder.getPieces().first().getNumar() / p2.getPieces().first().getNumar();  
 polinomToSubstract = *new* Polinom(String.*format*("%.2fx^%d", -division, difference));  
 polinomToSubstract = Polinom.*multiplication*(p2, polinomToSubstract);  
 remainder = Polinom.*addition*(remainder, polinomToSubstract);  
 *if* (!result.getPieces().add(*new* Monom(division, difference))) {  
 result.getPieces().last().setNumar(division);  
 }  
 *if* (remainder.getPieces().size() != 0 && (difference+p2.getPieces().first().getGrad() == remainder.getPieces().first().getGrad()) && remainder.getPieces().first().getNumar()<1) {  
 remainder.getPieces().remove(remainder.getPieces().first());  
 }  
 *if* (remainder.getPieces().size() != 0) {  
 difference = remainder.getPieces().first().getGrad() - p2.getPieces().first().getGrad();  
 }  
  
 }  
 *if* (result.getPieces().size() != 1 && result.getPieces().last().getNumar() == 0) {  
 result.getPieces().remove(*new* Monom());  
 }  
 *if* (remainder.getPieces().size() == 0) {  
 remainder = *new* Polinom("0");  
 }  
 list.add(result);  
 list.add(remainder);  
 *return* list;  
}

* Aflu diferența de grad dintre primul polinom și rest și împărțirea dintre coeficientul primului monom din primul polinom și coeficientul din primul monom din cel de-al doilea polinom. Creez un nou polinom cu gradul ca diferența calculată anterior și coeficientul tot cel calculat anterior și scad acest polinom din primul apoi îl adaug în lista polinomului care va reprezenta câtul. Restul va fi determinat de ceea ce rămâne în primul polinom după ce s-a ieșit din while.

### Metoda derivate

* Parcurg pe rând toate monoamele din polinom, înmulțesc coeficientul cu gradul, scad gradul cu 1 și adaug noul monom în polinomul rezultat.

### Metoda integrate

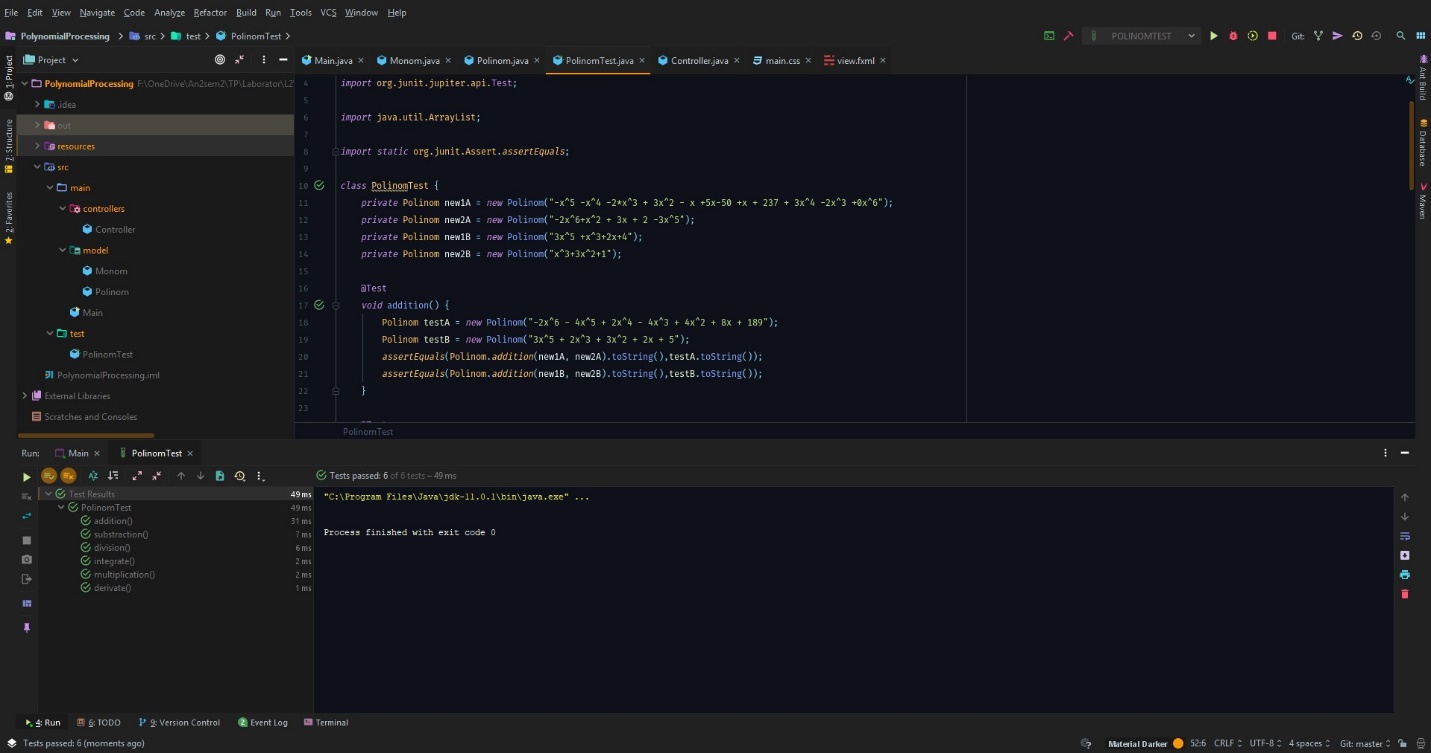
* Parcurg pe rând toate monoamele din polinom, împart coeficientul cu (gradul + 1), adun gradul cu 1 și adaug noul monom în polinomul rezultat.

# Implementare

* În cadrul acestui proiect am un număr de n clase: Main, care pornește aplicația JavaFX; Monom în care se stochează coeficientul și gradul unui monom; Polinom, aici am adăugat un TreeSet de Monom sortate descrescător după grad; Controller, aici avem implementate toate acțiunile necesare pentru a calcula fiecare operație; și clasa de test PolinomTest pe care o voi prezenta la capitolul de testare. View-ul este prezent în partea de resurse fiind reprezentat de fișierul view.fxml.
* Stilizarea se realizează cu ajutorul unui fișier main.css prin care se modifică proprietățile componentelor din view.
* Proiectul este structurat în 4 pachete: main, model, controllers și test.

# Testare

* În pachetul test am creat clasa PolinomTest.java în care am creat 4 polinoame, cu scopul de a testa operațiile. Apoi am creat metode de test pentru fiecare operație și am folosit 2 assertEquals pentru a verifica egalitatea între rezultatul operațiilor și rezultatele introduse manual.



# Concluzii

* Tema de față a fost o provocare pentru mine datorită faptului că am învățat un nou mod de proiectare a interfeței grafice (GUI) prin JavaFX. Astfel că am reușit să îmi însușesc cunoștințe de xml și css pe lângă cele de Java.
* Proiectul poate fi dezvoltat prin suportul de coeficienți reali și a puterilor negative. Ca și operații s-ar putea adăuga aflarea soluțiilor sau rezolvarea sistemului.

# Bibliografie

* Împărțirea polinoamelor: <https://www.youtube.com/watch?v=get8UJspyT4>
* Training JavaFX: https://teamtreehouse.com/home