**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**

**DEPARTAMENTUL CALCULATOARE**

**TEMA 1**

**Calculator de polinoame**

**Documentație**

**Nume: Litu Saviana**

**Grupa: 30221**

**An academic 2020 – 2021**

**Cuprins**

[**Obiectivul temei** 3](#_Toc66810739)

[**Analiza problemei, modelare, scenarii, cazuri de utilizare** 4](#_Toc66810740)

[**Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator)** 6](#_Toc66810741)

[**Implementare** 10](#_Toc66810742)

[**Rezultate** 20](#_Toc66810743)

[**Concluzii** 23](#_Toc66810744)

[**Bibliografie** 23](#_Toc66810745)

# **Obiectivul temei**

Obiectivul acestei teme este de a propune și implementa o aplicație Java în cadrul căreia să se realizeze un calculator de polinoame cu o singură variabilă și coeficienți întregi. Această aplicație va dispune și de o interfață grafică, în care utilizatorul poate introduce două polinoame, poate selecta operația care să se realizeze asupra lor și poate vedea rezultatul.

În vederea realizării acestui obiectiv principal, definim următoarele obiective secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv** | **Descriere** | **Capitol** |
| **Analiza problemei** | **Se analizează enunțul problemei și se dezvoltă un design inițial** | **2** |
| **Alegerea structurilor de date potrivite** | **Se aleg structuri de date care să faciliteze realizarea operațiilor cu polinoame** | **3** |
| **Implementarea** | **Se vor crea clase în concordanță cu design-ul dorit** | **3** |
| **Proiectarea algoritmilor** | **Se vor scrie algoritmi pentru a realiza operațiile asupra polinoamelor** | **3** |
| **Realizarea interfeței grafice** | **Se va realiza interfața utilizator utilizând JavaFX** | **4** |
| **Testarea** | **Se vor efectua teste pentru fiecare operație implementată** | **4** |

# **Analiza problemei, modelare, scenarii, cazuri de utilizare**

În general, un polinom este o expresie construită dintr-una sau mai multe variabile și constante. Termenii polinomului se numesc monoame, acestea la rândul lor fiind alcătuite dintr-o constantă, numită coeficient, înmulțită cu una sau mai multe variabile. Fiecare variabilă poate avea un exponent.

Analizând problema propusă, observăm încă din enunț că polinoamele cu care vom lucra sunt de o singura variabilă și au coeficienți întregi. Așadar, un exemplu de polinom corect definit care poate fi introdus de utilizator în interfața grafică poate fi: ”2x^3-10x^2+9x-5”.

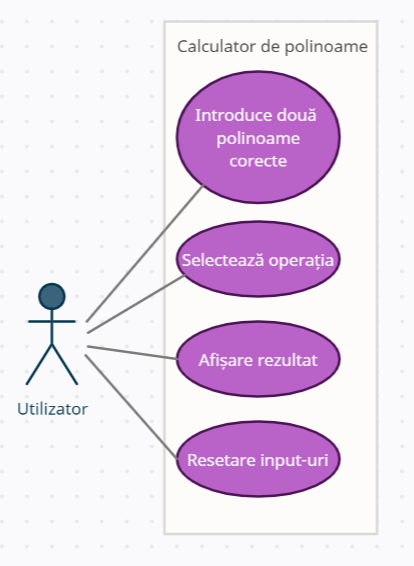
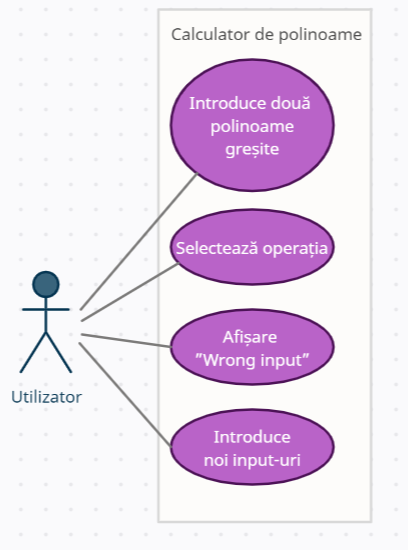
După ce utilizatorul introduce polinomul, acesta urmează să fie parsat, pentru a se putea reține coeficientul și exponentul fiecărui monom. Astfel, se vor putea realiza operațiile implementate asupra polinoamelor cu ușurință.

Așadar, în urma analizei, am luat următoarele decizii de modelare a aplicației: implementarea a doua clase – Polynomial și Monomial, prima având drept atribut un ArrayList de monoane, adica de tip Monomial, și o metodă parsePolynomial, prin intermediul căreia voi parsa fiecare polinom, folosind un Regex care separă fiecare coeficient și fiecare exponent în parte. Clasa Monomial va conține atribute întregi – coefficient și expoenent. Apoi, pentru a realiza metodele specifice operațiilor pe polinoame, creez clasa Operations, unde implementez adunarea, scăderea, înmulțirea și derivarea. Deoarece am decis să implementez doar aceste operații, coeficienții de tip întreg vor fi potriviți și nu va trebuie sa folosesc tipul Double sau Float.

Trecând la partea de modelare a interfeței utilizator, am ales ca utilizatorul sa aibă două field-uri unde să introducă polinoamele, fie de la tastatură, fie apăsând butoanele calculatorului. Apoi, pentru a alege operația dorită, utilizatorul va trebui sa apese un singur buton, care va conține denumirea operației, iar în urma apăsării acestuia, se va calcula și afișa rezultatul într-un al treilea field, care corespunde rezultatului, neputând fi editat. Totuși, neimplementând operațiile de împărțire și integrare, la apăsarea butoanelor corespunzătoare nu se va afișa un rezultat, ci un mesaj prin care utilizatorul este informat că aceste operații nu au fost implementate.

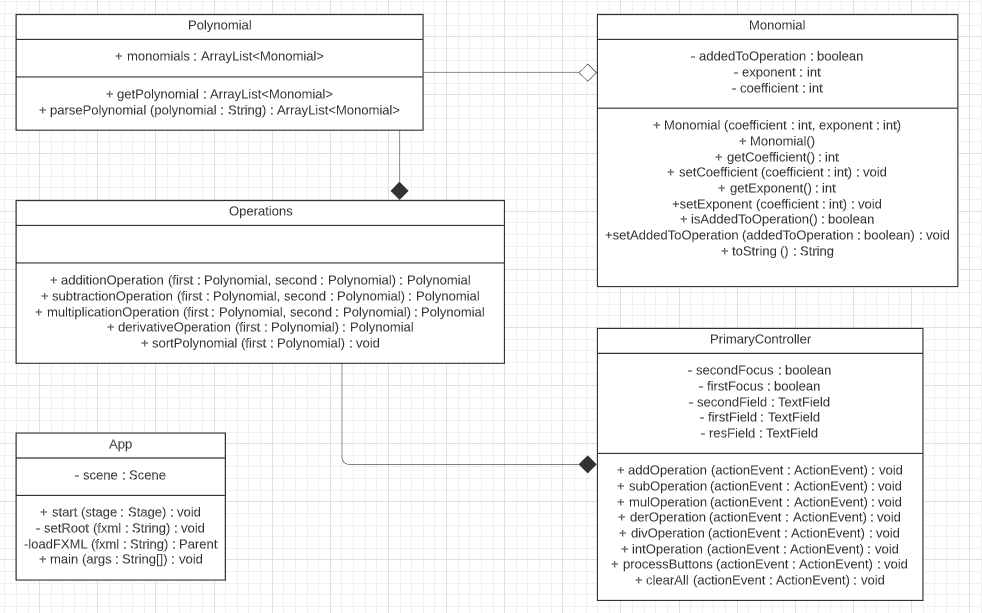
Rezultatul va fi tot un polinom, reprezentat sub o forma similară cu cea în care a fost introdus de utilizator. De asemenea, după obținerea rezultatului, utilizatorul poate folosi o nouă operație, trebuind doar să apese butonul DEL, care va goli toate field-urile. Apoi, se pot introduce noi polinoame și se poate alege o altă operație.

În continuare, voi prezenta cazul de utilizare ideal și un caz de utilizare alternativ, care apare în momentul în care utilizatorul introduce date greșite (de exemplu: introduce în primul field ”a” și în al doilea field ”b”).

1. **Cazul ideal**
2. Utilizatorul introduce, fie de la tastatură, fie folosind butoanele puse la dispoziție în interfață, pe calculator, două polinoame corect definite, în cele două field-uri corespunzătoare lor.
3. Utilizatorul alege operația dorită, apăsând butonul specific acesteia.
4. În field-ul corespunzător rezultatului va apărea polinomul obținut în urma aplicării operației asupra input-urilor.
5. Dacă utilizatorul dorește să efectueze o nouă operație, poate folosi butonul DEL, care va goli toate field-urile.
   1. În acest caz, pașii 1, 2 și 3 se vor repeta.
6. **Caz alternativ (de eroare)**
7. Utilizatorul introduce, fie de la tastatură, fie folosind butoanele puse la dispoziție în interfață, pe calculator, unul sau ambele input-uri într-un format greșit.
8. La apăsarea oricărui buton care corespunde unei operații (în afară de împărțire și integrare), în field-ul rezultatului se va afișa mesajul ”Wrong input”, iar field-urile care corespund celor doua polinoame de intrare vor fi golite automat.
9. Utilizatorul poate introduce noi polinoame.
   1. În acest caz, dacă input-urile vor fi scrise într-o formă greșită din nou, se vor repeta pașii 1 și 2 ai acestui caz.
   2. Dacă input-urile vor fi corecte, se va reveni la cazul ideal.

# **Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator)**

* **Diagrama UML**

****

* **Structura de date**

Se va folosi preponderent structura de date ArrayList, deoarece este rapidă, facilitează iterarea prin elemente și adăugarea de noi elemente, care se face by default la sfârșitul listei și poate conține duplicate.

* **Proiectarea claselor**

*Clasa App*: clasă în care se află metoda main, deci din care se rulează aplicația.

*Clasa Monomial*: clasa principală a acestui program, care are drept atribute coeficientul, exponentul, dar și addedToOperation, de tip boolean, prin intermediul căruia, la realizarea unei operații, se verifică dacă un monom anume a fost adăugat la rezultat sau nu, în urma comparării exponenților celor două polinoame implicate în operație. Această clasă cuprinde doi constructori, unul care nu primește parametri, dar și unul care primește coeficientul și exponentul, getters și setters pentru atribute, dar și metoda toString, realizată pentru a putea afișa monoamele sub formă de string-uri.

*Clasa Polynomial*: clasă care are drept atribut un ArrayList de Monomial, deoarece fiecare polinom va fi format din monoame. Metodele implementate sunt getPolynomial, care returneaza lista de monoame și parsePolynomial, care primește un string, iar în funcție de un Regex, separă coeficienții si exponenții fiecărui monom, returnând un ArrayList de monoame.

*Clasa Operations*: clasă în care am implementat operațiile pe polinoame, respectiv additionOperation, subtractionOperation, multiplicationOperation, derivativeOperation. De asemenea, am mai scris și metoda sortPolynomial, pentru a sorta monoamele obținute în urma aplicării operațiilor, pentru a obține un polinom sub forma definită ca fiind corectă, adică cea în care exponenții sunt în ordine descrescătoare.

*Clasa PrimaryController*: clasă în care implementez funcționalitatea butoanelor folosite în interfața grafică, dar și a TextField-urilor create.

* **Algoritmi**

Algoritmii folosiți pentru realizarea operațiilor asupra polinoamelor sunt implementați în clasa Operations:

* **Algortim pentru adunare**

Se regăsește în metoda additionOperation, primește două polinoame ca input și returnează tot un polinom. Aici creez un nou polinom, în care voi reține suma și pe care îl voi returna. Folosesc două foreach-uri, cu ajutorul cărora iterez prin monoamele care se află în ArrayList-ul de Monomial al fiecarui polinom implicat în aceasta operație, ca input. Apoi, verific, pe rând, dacă monomul primul polinom are acelasi exponent cu monomul celui de-al doilea polinom. Dacă au același exponent, atunci creez un nou monom pe care îl rețin în ArrayList-ul de monoame al rezultatului și îi setez exponentul – egal cu exponentul unuia dintre cele două monoame verificate, iar coeficientul – suma coeficienților celor două monoame. De asemenea, pentru toate monoamele care au fost adunate cu succes, atributul addedToOperation va lua valoarea true.

Totuși, ar putea exista și monoame cu exponent diferit. Pentru acest caz, mai scriu câte un foreach pentru fiecare listă de monoame a celor două polinoame introduse, pentru a verifica dacă monoamele conținute de acestea au atributul addedToOperation true sau false. Daca acesta este false, atunci monomul respectiv este adăugat în lista de monoame a rezultatului.

* **Algoritm pentru scădere**

Se regăsește în metoda subtractionOperation, primește două polinoame ca input și returnează tot un polinom. Aici creez un nou polinom, în care voi reține diferența și pe care îl voi returna. Folosesc două foreach-uri, cu ajutorul cărora iterez prin monoamele care se află în ArrayList-ul de Monomial al fiecarui polinom implicat în aceasta operație, ca input. Apoi, verific, pe rând, dacă monomul primul polinom are acelasi exponent cu monomul celui de-al doilea polinom. Dacă au același exponent, atunci creez un nou monom pe care îl rețin în ArrayList-ul de monoame al rezultatului și îi setez exponentul – egal cu exponentul unuia dintre cele două monoame verificate, iar coeficientul – diferența dintre coeficientul primului monom si coeficientul celui de-al doilea monom. De asemenea, pentru toate monoamele care au fost scăzute cu succes, atributul addedToOperation va lua valoarea true.

Totuși, ar putea exista și monoame cu exponent diferit. Pentru acest caz, mai scriu câte un foreach pentru fiecare listă de monoame a celor două polinoame introduse, pentru a verifica dacă monoamele conținute de acestea au atributul addedToOperation true sau false. Daca acesta este false, atunci monomul respectiv este adăugat în lista de monoame a rezultatului.

* **Algoritm pentru înmulțire**

Se regăsește în metoda multiplicationOperation, primește două polinoame ca input și returnează tot un polinom. Aici creez un nou polinom, în care voi reține rezultatul înmulțirii și pe care îl voi returna. Folosesc două foreach-uri, cu ajutorul cărora iterez prin monoamele care se află în ArrayList-ul de Monomial al fiecarui polinom implicat în aceasta operație, ca input. Apoi, creez un nou monom pe care îl rețin în ArrayList-ul de monoame al rezultatului și îi setez exponentul – egal suma exponenților celor două monoame, iar coeficientul – coeficientul primului monom înmulțit cu coeficientul celui de-al doilea monom.

* **Algoritm pentru derivare**

Se regăsește în metoda derivativeOperation, primește un polinom și returnează tot un polinom. Aici creez un nou polinom, în care voi reține derivata și pe care îl voi returna. Folosesc un foreach, cu ajutorul căruia iterez prin monoamele care se află în ArrayList-ul de Monomial al polinomului primit ca input. Creez un nou monom pe care îl rețin în ArrayList-ul de monoame al rezultatului și îi setez exponentul – egal cu exponentul monomului verificat în foreach - 1, iar coeficientul – coeficientul monomului verificat înmulțit cu exponentul acestuia.

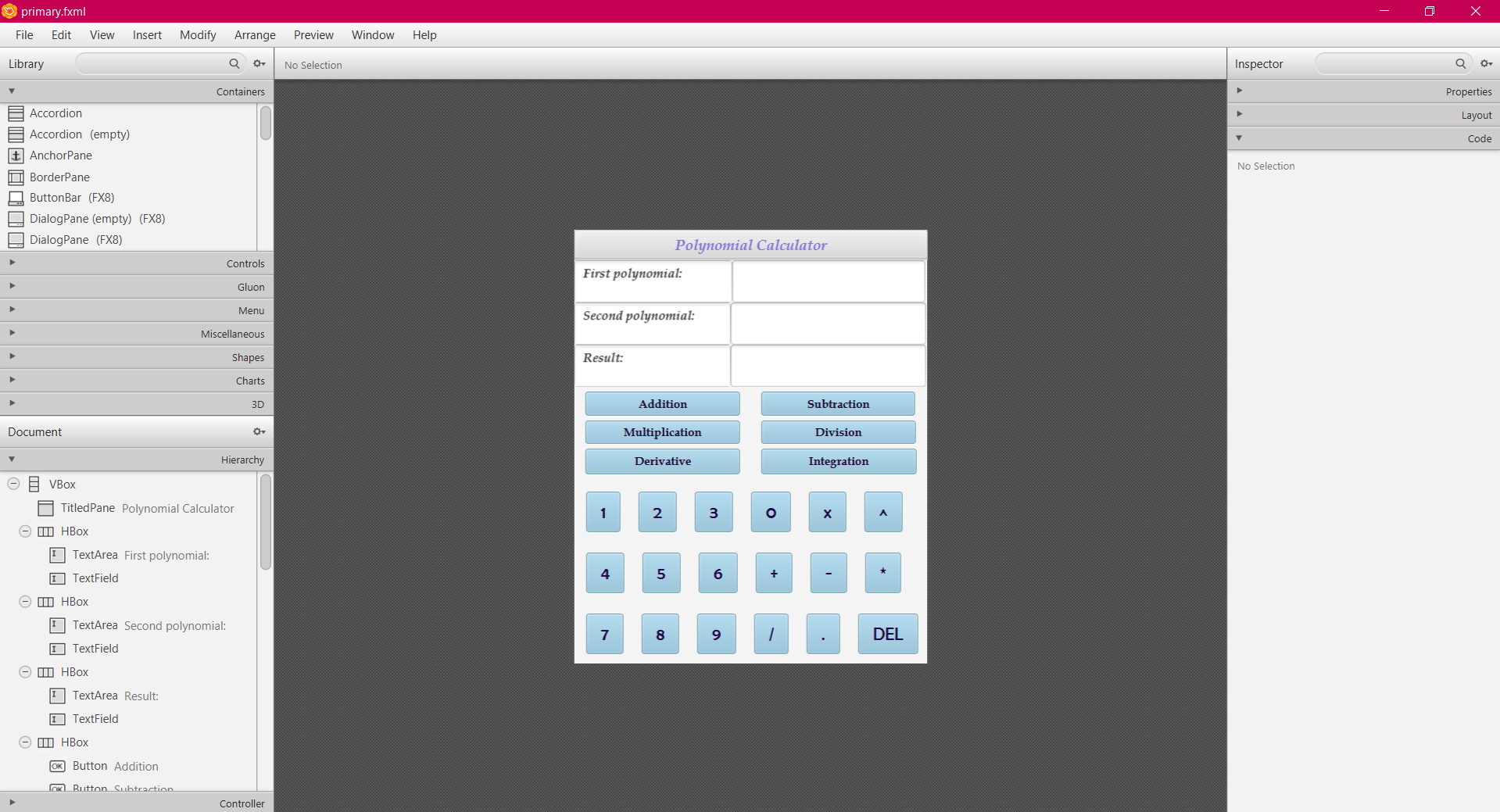
* **Algoritm pentru sortarea polinoamelor**

Se regăsește în metoda sortPolynomial, primește un polinom și este de tip void. Aici folosesc un boolean numit monomialParsed, care inițial este fals. Cât timp acesta este fals, va deveni true, iar într-un for simplu prin monoamele polinomului verific daca exponentul unul monom aflat la index este mai mic decat exponentul urmatorului monom. Dacă este mai mic, atunci interschimb monoamele între ele, folosind un monom auxiliar, iar monomialParsed devine din nou false. Astfel, reușesc să verific dacă toate monoamele dintr-un polinom se află în ordinea bună – ordinea descrescătoare a exponenților.

O altă metodă importantă este implementată în clasa Monomial, și anume toString, cea prin intermediul cărora ”traduc” un monom într-un string. Aici, în funcție de valoarea exponentului și a coeficientului, decid ce să afișez.

* **Interfața utilizator**

Interfața a fost primul aspect al acestui proiect pe care l-am realizat. Mi-am creat un proiect Java de tip Maven și am dezvoltat interfața grafică folosind JavaFX, mai exact partea de SceneBuilder. Cu ajutorul acesteia mi-am modelat interfața, alegând si editând cadranul in care sa se afle toate TextField-urile si butoanele, prin drag and drop.

****

În acest screenshot se poate vedea cum arata mediul în care mi-am realizat interfața utilizator.

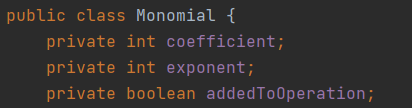
Fiecare TextField folosit are în secțiunea Code un fx:id, căruia îi dau un nume, apoi îl declar în clasa PrimaryController, cu o adnotație @FXML, pentru a-l putea utiliza și lucra cu el în cadrul metodelor care țin de operații.

Butoanele pentru operații au câte o metodă separată, în clasa PrimaryController, fiind precedate de adnotații @FXML pentru a fi recunoscute. Restul butoanelor se încadrează în aceeași metodă din PrimaryController, și anume processButtons, care verifică în ce TextField se apasă click cu mouse-ul, apoi scrie cifra, respectiv semnul care corespunde butonului apăsat de utilizator.

# **Implementare**

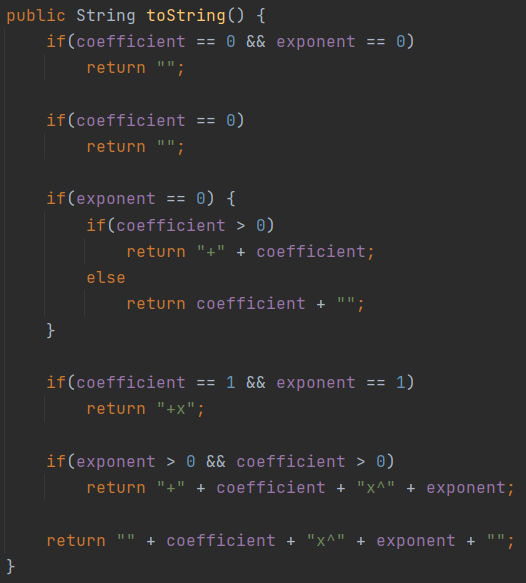
Voi începe prin a prezenta clasa *Monomial*, care este nucleul acestui proiect, fiind unitatea de bază de la care se pornește, un polinom fiind alcătuit din monoame.

Această clasă are atributele coefficient(int), exponent(int) și addedToOperation(boolean).

****

Rolul primelor două atribute este clar – de a stoca coeficientul și exponentul unui monom, iar cel de-al treilea a fost folosit pentru a verifica dacă un anume monom a fost folosit sau nu în cadrul operațiilor de adunare și scădere.

Fiecare atribut are metode de setter și getter, iar pe lângă acestea, am scris și o metodă toString, fără parametri și care returneaza un String, prin intermediul căreia voi afișa monoamele ca string-uri.



Continuăm cu prezentarea clasei *Polynomial*. Aceasta clasă ne va ajuta să stocăm polinoamele și să le parsăm, în vederea separării și reținerii coeficienților și a exponenților, pentru a îi putea folosi în cadrul operațiilor cu polinoame.

Atributul clasei este monomials, de tip ArrayList<Monomial>, deoarece fiecare polinom va avea lista sa de monoame din care este alcătuit.

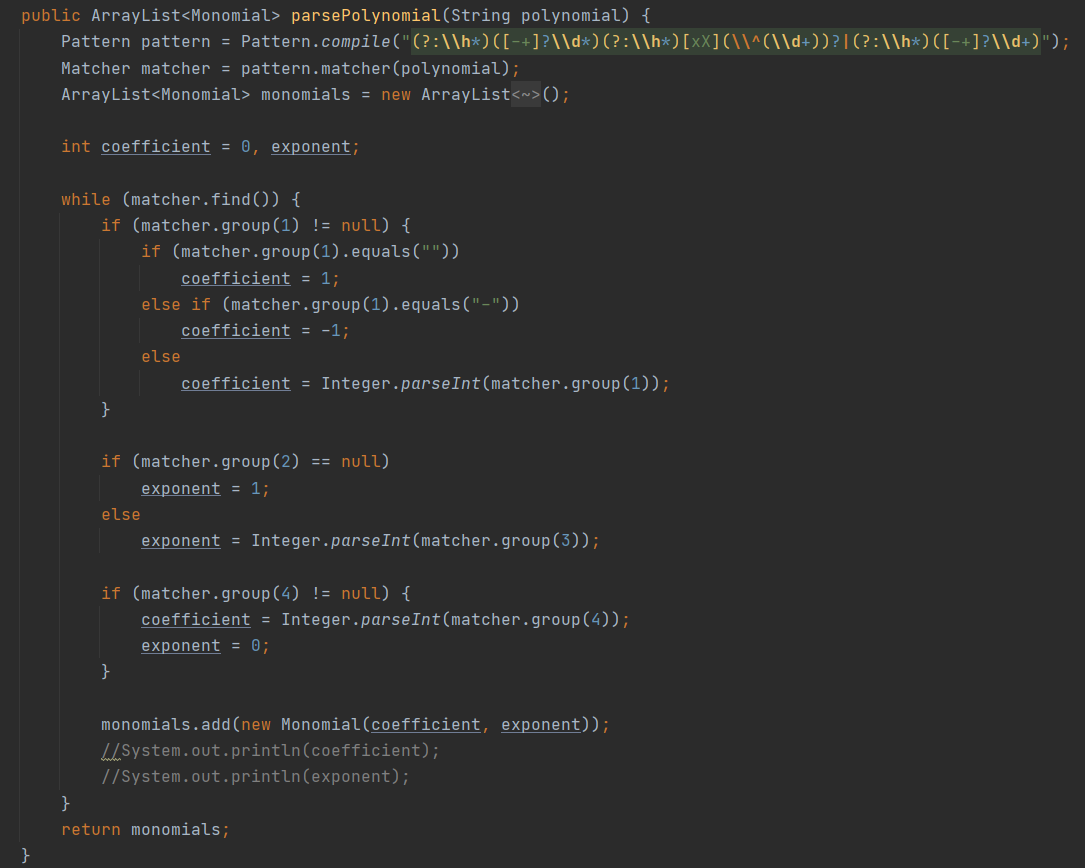
****

Singura metodă implementată în aceasta clasă este parsePolynomial, care primește un string și returnează ArrayList<Monomial>.

Pentru parsarea polinoamelor am folosit un Regex:

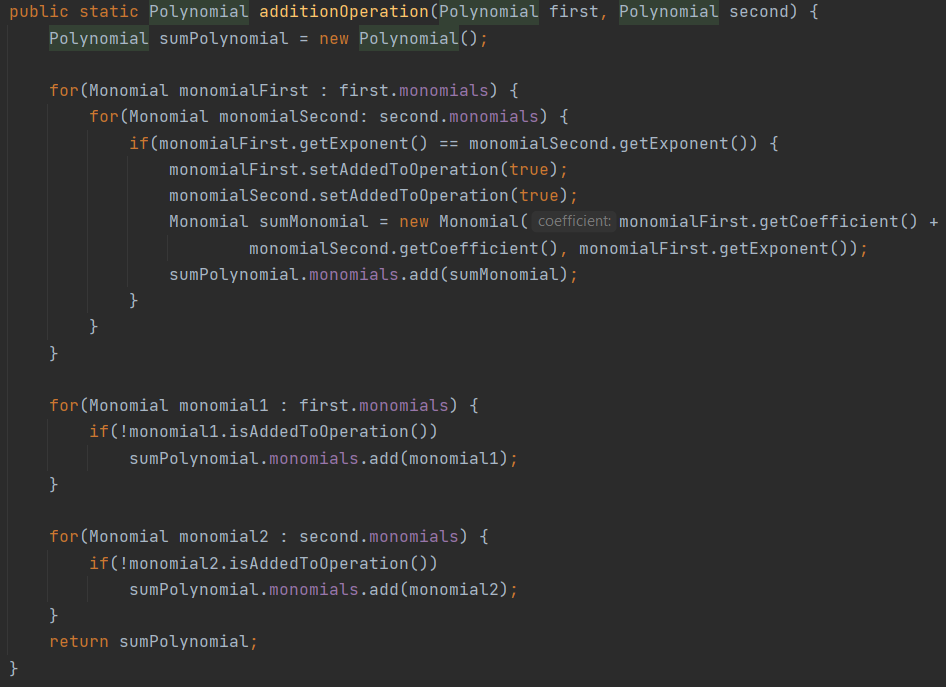
**"(?:\\h\*)([-+]?\\d\*)(?:\\h\*)[xX](\\^(\\d+))?|(?:\\h\*)([-+]?\\d+)"**

Acesta este preluat de pe StackOverflow și l-am testat pe site-ul regex101.com, unde am făcut mai multe teste, cu diferite polinoame. Am observat următoarele: grupul 1 obținut este întotdeauna coeficientul, grupul 2 este reprezentat de semnul ”^” și exponent, grupul 3 conține doar exponentul, iar grupul 4 apare doar atunci când avem doar coeficient, deci expoenentul este 0. Așadar, în funcție de aceste aspecte, am realizat funcția de parsare, în care am verificat, pe rând, fiecare grup obținut, în cazul în care între polinom și Regex există o potrivire.

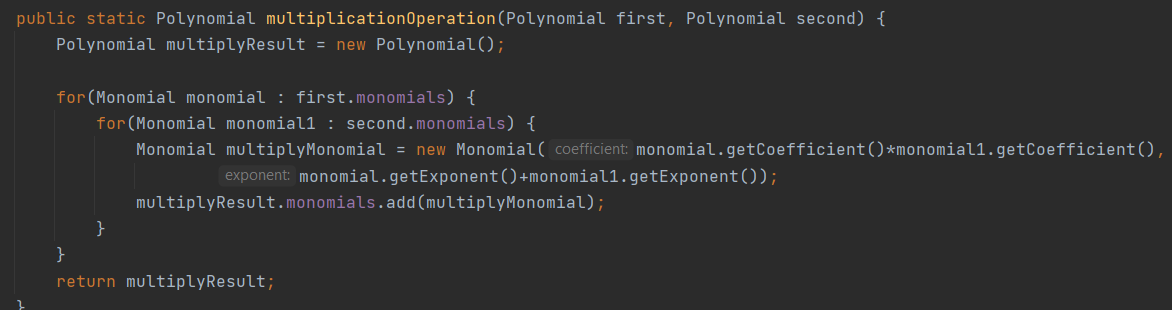
****

Clasa *Operations* este clasa în care am implementat operațiile de adunare, scădere, înmulțire și derivare. Aici nu am atribute, ci doar metode. Metodele sunt următoarele: additionOperation, subtractionOperation, multiplicationOperation și derivativeOperation.

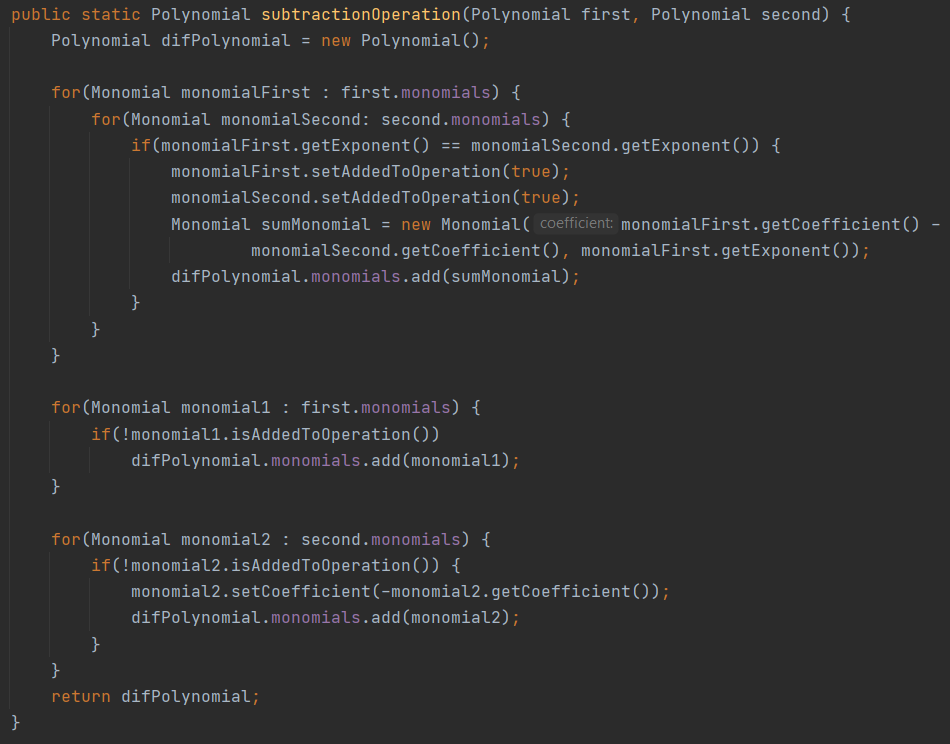
Încep cu prezentarea metodei pentru adunare. Aceasta primește două polinoame și returnează tot un polinom. Funcționează întocmai după cum am explicat algoritmul în capitolul anterior.



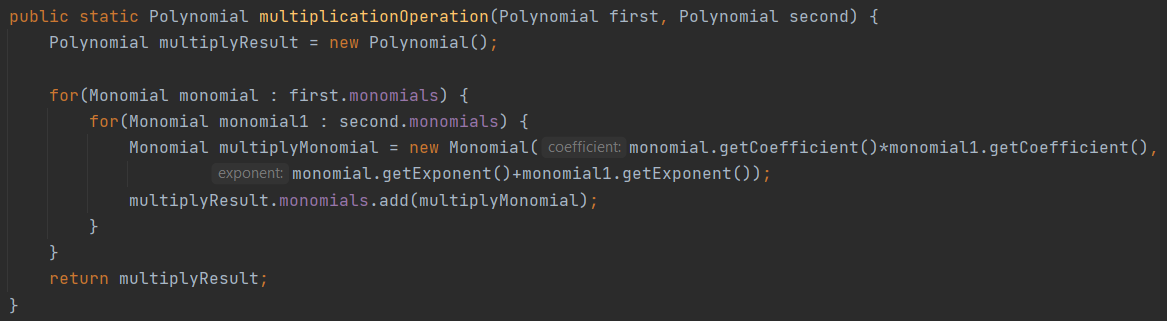
La fel și metoda pentru înmulțire, primește două polinoame, returnează un rezultat sub formă de polinom și respectă algoritmul descris.



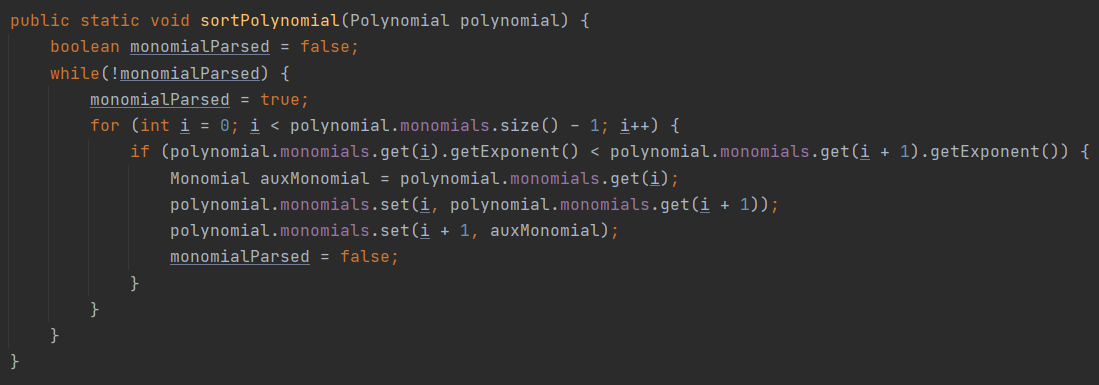
Metoda pentru scădere, la fel ca adunarea, primește două polinoame, returnează un polinom și este practic implementarea algoritmului descris de mine mai sus.



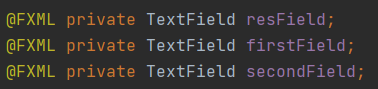
Ultima metodă, ce ține de operațiile cu polinoame, este derivata. Aceasta primește un singur polinom și returnează derivata sa, sub formă de polinom. Din nou, respectă algortimul pe care l-am detaliat mai sus.



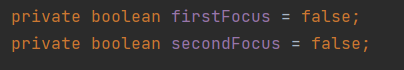
O metodă adițională de care am avut nevoie pentru a sorta rezultatele obținute, în ceea ce privește ordinea monoamelor, în funcție de exponent, este sortPolynomial, care primește un polinom și este de tip void, dar are ca efect ordonarea monoamelor unui polinom în ordinea descrescătoare a epxonenților.



Trecând la următoarea clasă – avem *PrimaryController*. Aici gestionez funcționalitatea butoanelor și a TextField-urilor pe care le-am implementat în interfața grafică. După cum am povestit anterior, folosesc adnotația @FXML înaintea unui TextField și înaintea metodelor prin care controlez butoanele.



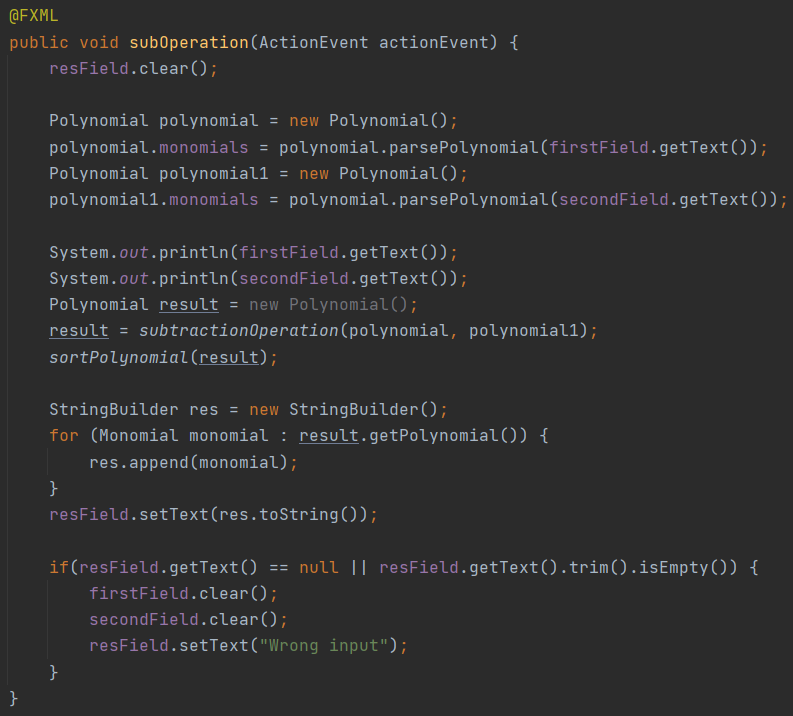
Pe lângă TextField-uri, mai folosesc și două variabile de tip boolean, anume firstFocus și secondFocus, cu ajutorul cărora verific care TextField este selectat atunci când implementez scrierea folosind butoanele de care dispune calculatorul. Dacă firstFocus este true și secondFocus este false, atunci se va scrie primul polinom, iar dacă secondFocus este true și firstFocus este true, se va scrie al doilea polinom.



Metoda prin care controlez operația de adunare este numită addOperation, având parametrul actionEvent. Așadar, butonul pentru adunare va funcționa atunci când este apăsat astfel: creez două obiecte noi de tip Polynomial, deci polinoamele pe care le va introduce utilizatorul. Apoi folosesc funcția de parsare asupra ambelor polinoame, deci se va returna câte o listă de monoame pentru fiecare dintre cele două, deoarece așa e definită metoda, deci voi reține aceste două liste în atributul monomials al fiecărui polinom. Apoi mai creez un polinom în care să rețin rezultatul și apelez metoda de adunare din clasa Operations și sortez rezultatul obținut. Pentru a putea adăuga monoamele rezultatului pentru afișare am folosit tipul StringBuilder, deoarece dispune de metoda append. În final, setez textul din field-ul care corespunde primului polinom să fie chiar string-ul obținut după apelarea metodei toString. De asemenea, dacă input-urile nu sunt corecte, field-ul pentru rezultat va fi gol, așa că verific dacă acesta este null, iar dacă este, dau clear input-urilor și afișez un mesaj de eroare – ”Wrong input”.



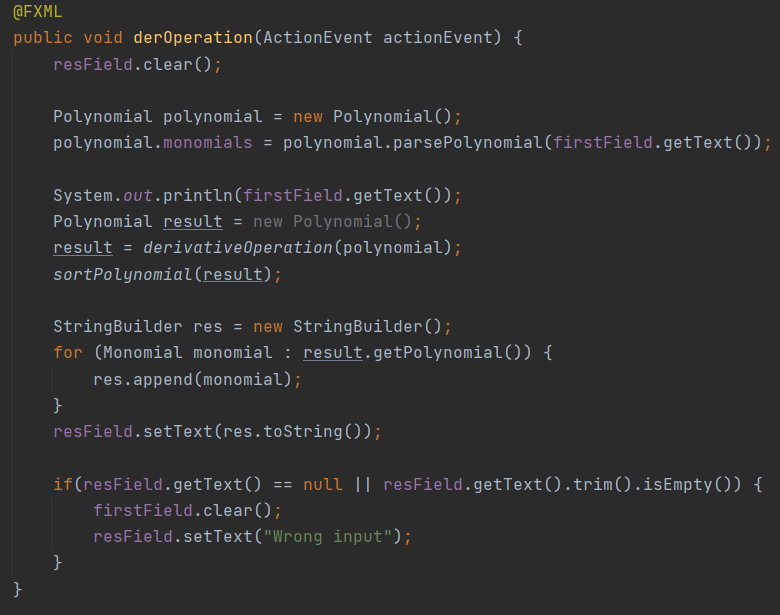
Metoda prin care controlez operația de scădere este numită subOperation, având parametrul actionEvent. Aceasta este structurată la fel ca adunarea și funcționează la fel. Desigur, singura diferență este că apelez metoda care corespunde scăderii, din clasa Operations.



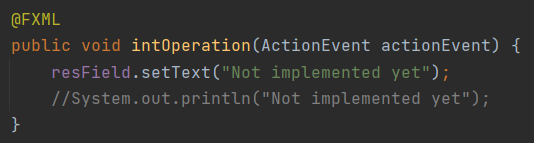
Metoda pentru butonul de înmulțire este următoarea, asemănătoare cu cele de dinainte:

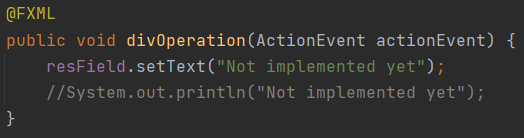


Metoda pentru butonul de derivare este aceasta:



De asemenea, chiar dacă nu am implementat efectiv metode pentru a realiza împărțirea și integrarea, în clasa Operations, am configurat butoanele astfel încât să informeze utilizatorul ca aceste operații nu se pot realiza deocamdată.

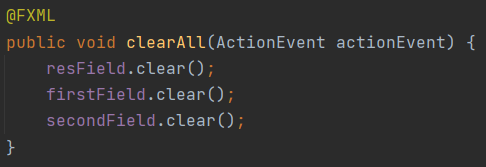




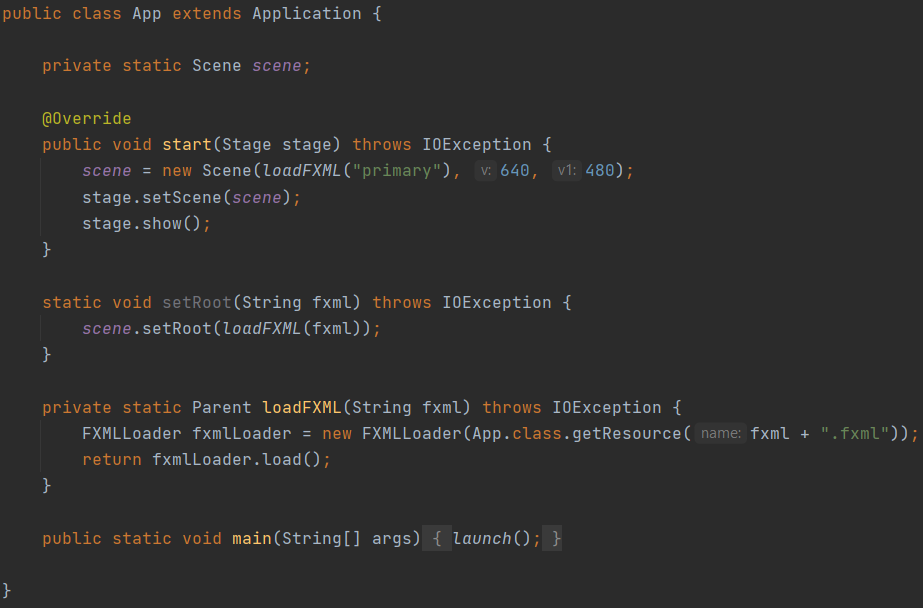
O altă metodă este processButtons, care se ocupă de restul butoanelor. Aici adaug câte un EventHandler pentru fiecare dintre cel două field-uri în care se introduc polinoamele, care funcționează atunci când se dă click cu mouse-ul pe field-ul dorit. Pentru primul field, boolean-ul firstFocus devine true, iar secondFocus este false, iar în cazul celui de-al doilea field, secondFocus este true și firstFocus este false. Apoi verific care dintre field-uri este focusat și aleg să se scrie în el.



Ultima metodă este clearAll, cu ajutorul căreia golesc field-urile atunci când apăs tasta DEL, înainte de efectuarea unei noi operații sau în cazul în care vreau să fac o schimbare la input-uri.



Ultima clasă este clasa App, care este Main-ul acestui program. Aceasta a fost creată automat atunci când mi-am setat dependințele pentru JavaFX, iar de aici rulez aplicația.



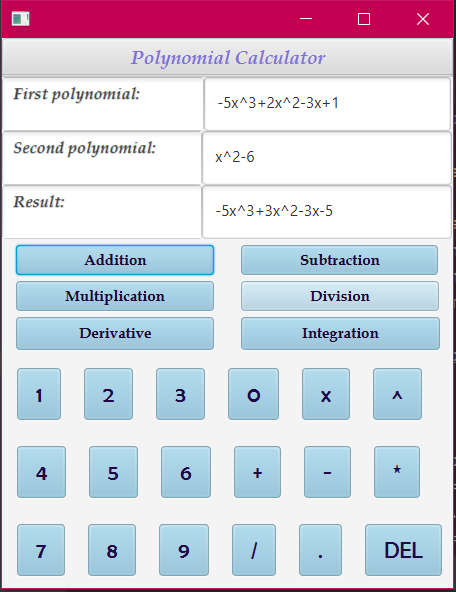
# **Rezultate**

Testarea unitară s-a impus în ultima perioadă în dezvoltarea proiectelor scrise în limbajul Java şi numai, pe măsura apariţiei unor utilitare gratuite de testare a claselor, care au contribuit la creşterea vitezei de programare şi la micşorarea drastică a numărului de bug-uri. Cel mai folosit utilitar pentru testarea unitară a claselor Java este JUnit.

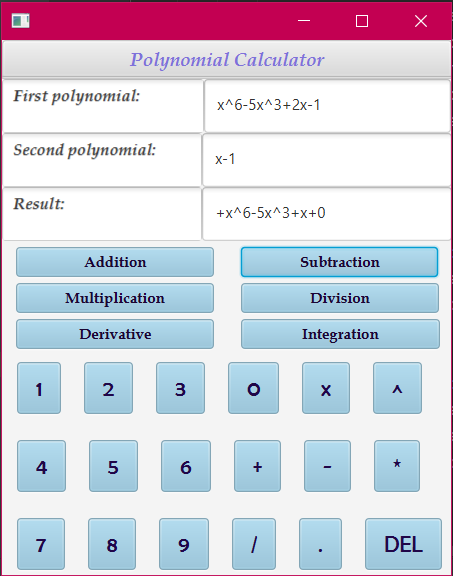
În cadrul acestei teme nu am folosit testarea unitară cu JUnit, ci doar m-am gândit la mai multe posibile input-uri și am testat direct în interfața grafică, pentru a vedea dacă voi obține un rezultat corect.

Voi atașa câteva screenshot-uri cu teste făcute de mine.

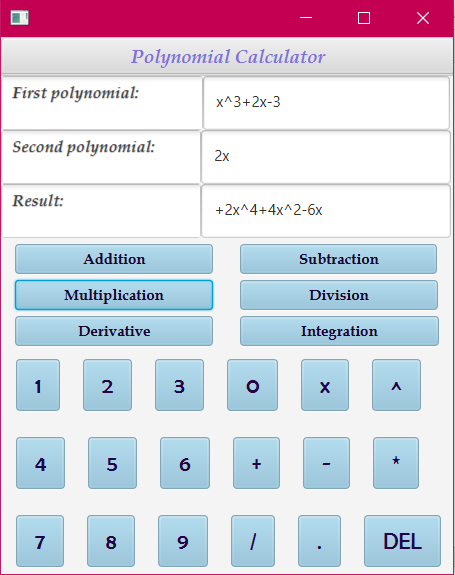
* Test pentru operația de adunare



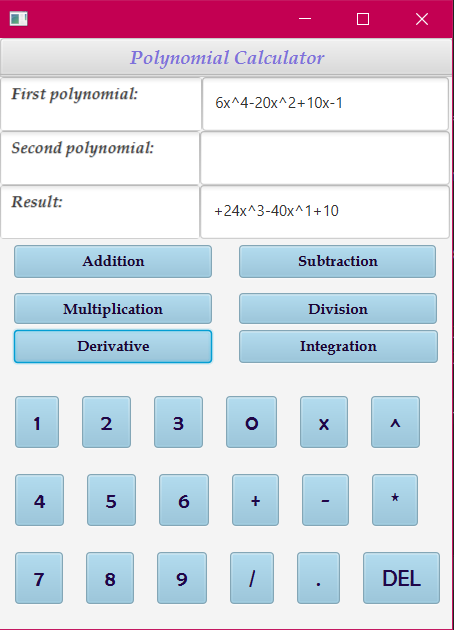
* Test pentru operația de scădere



* Test pentru operația de înmulțire



* Test pentru operația de derivare



# **Concluzii**

În concluzie, această temă a fost, din punctul meu de vedere, o provocare, în urma căreia am învățat cum să lucrez mai bine cu interfețele utilizator, mai exact cum să folosesc JavaFX, în combinație cu un proiect de tip Maven. Aceasta a fost partea la care mi-a făcut plăcere să lucrez, folosindu-mă de SceneBuilder și puțină creativitate. De asemenea, datorită cerințelor impuse am aprofundat și subiectul legat de Regex și am înțeles mai bine fenomenul și câteva dintre regulile în funcție de care se scriu aceste expresii. Așadar, mi s-a părut o temă potrivită pentru aprofundarea unui limbaj de programare orientat pe obiecte.

Ca dezvoltări ulterioare, m-am gândit la implementarea celorlalte două operații și la studierea problemelor pe care le-am întâmpinat atunci când am vrut să împart programul în pachete, astfel încât să pot ajunge să urmez modelul arhitectural. De asemenea, aș reface interfața grafică, pentru că am greșit atunci când am început să o implementez, deoarece nu am setat ca fiecare buton și TextField să se muleze în funcție de părinte. O altă schimbare ar consta în îmbunătățirea funcției mele de toString, pentru a afișa polinoamele într-un mod mai natural, mai apropiat de realitate și implementarea unei funcții cu ajutorul căreia rezultatul să fie reprezentat într-o formă corectă în toate cazurile posibile. În prezent, există cazuri în care rezultatul meu are termeni cu acelasi exponent, dar ai căror coeficienți nu se adună/scad.

# **Bibliografie**

<https://www.youtube.com>

<https://www.geeksforgeeks.org>

<https://stackoverflow.com>

<https://openjfx.io/openjfx-docs/>

<https://docs.oracle.com/>