Tehnici de programare

Tema 1

Procesare polinoame

Student : Bolba Raluca

Grupa : 30225

Cuprins

1. Obiectivul temei

Enunțul temei : Propuneți, proiectați și implementați un sistem de procesare a polinoamelor de o singură variabilă cu coeficienți întregi.

Obiectivul acestei teme este de a crea un sistem de procesare a polinoamelor de o singură variabilă cu coeficienți întregi care permite operații asupra unui singur polinom precum: găsirea valorii polinomului într-un punct dat, găsirea valorii derivatei polinomului într-un punct dat, găsirea derivatei polinomului și calculul rădăcinilor reale ale acestuia . De asemenea acest procesor de polinoame trebuie să asigure și implementarea unor operații pe un set de două polinoame, acestea fiind cele uzuale de adunare, scădere, înmulțire și împărțire. Pentru ca utilizatorul să interacționeze cât mai ușor cu acest sistem, am ales crearea unei interfețe grafice simplă și intuitivă, care oferă acces la toate operațiile descrise mai sus.

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

Una dintre operațiile ce trebuiesc efectuate în cadrul acestui sistem de procesare polinoamelor o reprezintă aflarea rădăcinilor reale ale unui polinom dat. Întrucât polinomul dat poate avea orice grad, iar pentru un grad al polinomului mai mic sau egal decât 3 se pot determina mai exact rădăcinile polinomului, trebuiesc stabilite niște cazuri particulare, și anume : găsirea rădăcinilor reale ale unui polinom de grad I, de grad II și de grad III. Aceste rădăcini le aflăm conform următoarelor:

I. Polinom de gradul întâi

ax+b=0

Unica rădăcină a acestui polinom este :

$$x = -\frac{b}{a}$$

II. Polinom de gradul doi

$ax^2+bx+c=0$
$\Delta = b^2-4ac$

- Când $\Delta > 0$ rădăcinile ecuației de grad 2 sunt reale și diferite și sunt date de formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Când $\Delta = 0$ rădăcinile ecuației de grad 2 sunt reale și egale și sunt date de formula:

$$x1 = x2 = -\frac{b}{2a}$$

- Când $\Delta < 0$ rădăcinile ecuației de grad 2 sunt complexe și diferite.

III. Polinom de gradul trei

$ax^3+bx^2+cx+d=0$

Pentru calcularea numărului de rădăcini reale și a rădăcinilor acestui polinom trebuie calculate următoarele valori:

$$f = \frac{\frac{3c}{a} - \frac{b^2}{a^2}}{3} \qquad g = \frac{\frac{2b^3}{a^3} - \frac{9bc}{a^2} + \frac{27d}{a}}{27} \qquad h = \frac{g^2}{4} + \frac{f^3}{27}$$

- Atunci când $h>0$ există o singură rădăcină reală iar aceasta se calculeaza conform urmatoarelor:

$$R = -\frac{g}{2} + \sqrt{h} \qquad S = \sqrt[3]{R} \qquad T = -\frac{g}{2} - \sqrt{h} \qquad U = \sqrt[3]{T}$$

$$x = (S + U) - \frac{b}{3a}$$

- În cazul special, când $f=g=h=0$, toate rădăcinile sunt reale și egale cu:

$$x1 = x2 = x3 = -\sqrt[3]{\frac{d}{a}}$$

- Atunci când *h<0* există trei rădăcini reale și diferite și se calculează conform urmatoarelor :

$$i = \sqrt{\frac{g^2}{4} - h} \qquad j = \sqrt[3]{i} \qquad k = \cos^{-1}\left(-\frac{g}{2i}\right)$$

$$L = -j \qquad M = \cos\frac{k}{3} \qquad N = \sqrt{3} * \sin\frac{k}{3} \qquad P = -\frac{b}{3a}$$

$$x1 = 2j * M + P$$
$$x2 = L(M + N) + P$$
$$x3 = L(M - N) + P$$

Pentru cazul general, când gradul polinomului pentru care se dorește aflarea rădăcinilor este mai mare decât trei, se utilizează *metoda lui Newton*. Rădăcina polinomului furnizată de această metodă este una aproximativă, care depinde de un punct de start, dat de utilizator. Metoda lui Newton constă în:

Având o funcție reală f, iar derivata ei, f ', vom începe cu stabilirea unei valori inițiale pentru x_0 pentru o rădăcină a funcției f. O aproximare mai bună pentru rădăcina funcției este

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)},$$

Geometric, (x_1, 0) este la intersecția cu axa x a tangentei funcției f în punctul (x_0). Procesul se repetă

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

până se atinge o valoare suficient de precisă.

Vom începe procesul cu o valoare inițială arbitrară x_0.

Întrucât polinoamele procesate se impune să aibă coeficienți întregi, pot apărea probleme atunci când trebuie efectuată împărțirea a două polinoame, deoarece unii coeficienți ai rezultatului pot să fie reali. În cazul celorlalte operații nu avem probleme, pentru ca nu apar astfel de constrângeri la tipul coeficienților.

Pentru a efectua oricare din operațiile dorite, trebuie stabilita o modalitate de modelare a datelor introduse de utilizator astfel încât ele să poată fi folosite ca date de intrare a sistemului de procesare a polinoamelor. Mai concret, trebuie stabilit sub ce formă introduce de la tastatură utilizatorul polinomul și cum este el procesat pe mai departe. În această privință, am ales ca introducerea polinomui să se facă prin introducerea coeficienților polinomului, despărțiți de virgulă. Șirul de caractere introdus este „spart" și transformat într-un vector de numere întregi, reprezentând coeficienții polinomului, ordonați astfel:

Dacă P = $a_0 X^n + \ldots + a_{n-2} X^2 + a_{n-1} X + a_n$

Atunci vectorul de coeficienti va fi : $a_0, a_1, \ldots, a_n$

Așadar, pentru rezolvarea acestei probleme trebuie luate în considerare procesarea datelor introduse de utilizator, determinarea datelor de intrare pentru sistemul de procesare a polinoamelor, modelarea tuturor operațiilor conform constrângerilor menționate mai sus și afișarea rezultatelor.

3. Proiectare (diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfață utilizator)

3.1 Proiectare clase

Pentru modelarea acestei probleme, am considerat că este necesară implementarea următoarelor clase : *Polinom, PolinomGrad1, PolinomGrad2, PolinomGrad3* și *InterfataUtilizator.*

Clasa *Polinom* are ca atribute *gradul* polinomului (grad) și *coeficienții* polinomului (coef). Această clasă modelează conceptul de polinom general, iar obiectele acestei clase sunt inițializate în cadrul constructorului cu același nume, care primește ca parametrii o valoare întregă reprezentând gradul polinomului și un tablou de valori întregi reprezentând coeficienții polinomului.

Clasa *PolinomGrad1* extinde clasa *Polinom*, moștenind atributele și metodele acesteia. Această clasă modelează un caz particular de polinom, anume cel de gradul întâi, iar obiectele acestei clase sunt inițializate în cadrul constructorului cu același nume, care nu face decât să apeleze constructorul *super(int grad, int[] coef)* a clasei părinte, parametrii pentru acesta fiind determinați în funcție de parametrii primiți de constructorul curent. Introducerea acestei clase a fost necesară pentru a supraîncărca metoda de determinare a rădăcinilor reale a polinomului,

astfel încât aceasta să furnizeze o valoare exactă și nu una aproximată. Metoda supraîncărcată este `solve()`.

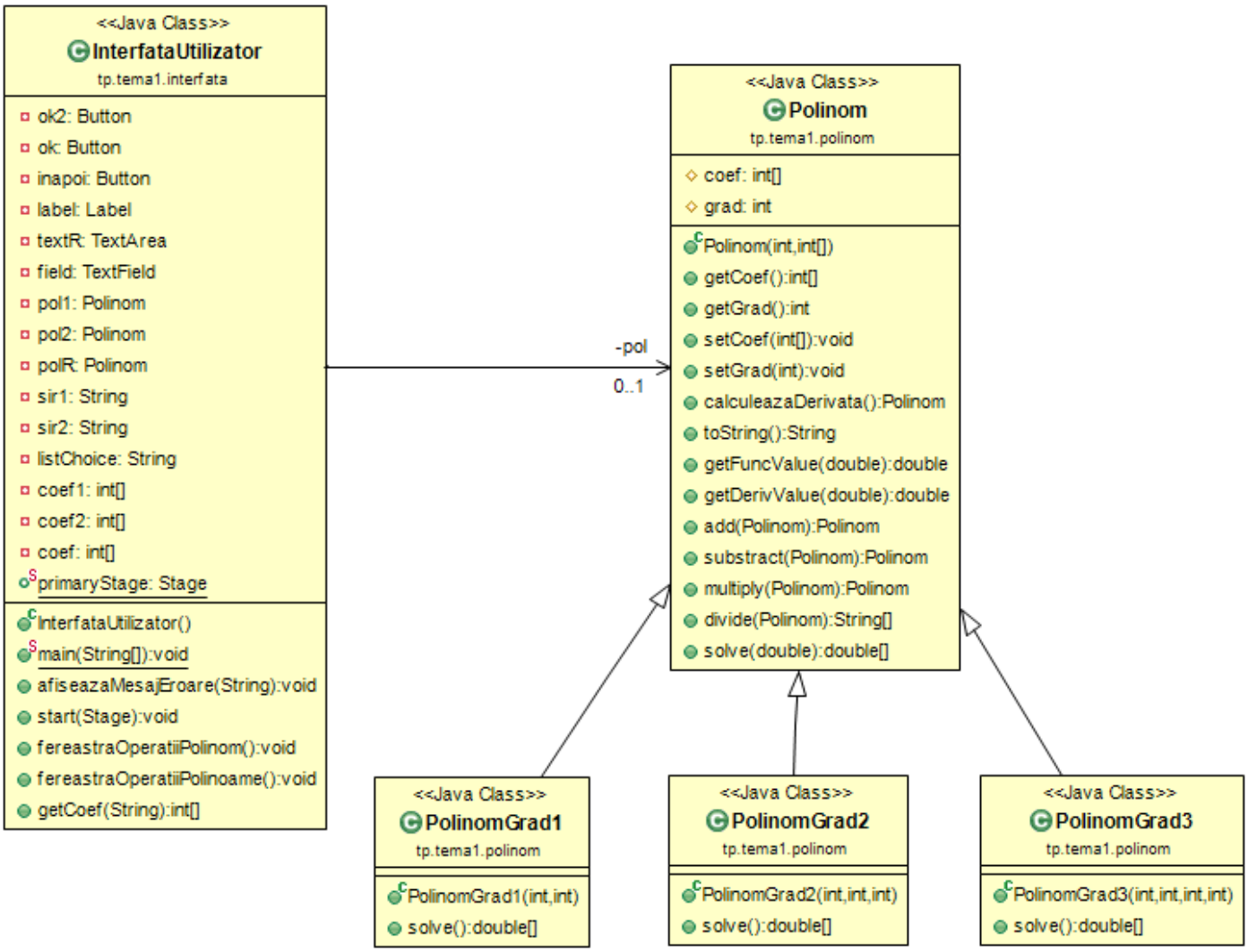
Clasa *PolinomGrad2* extinde și ea clasa *Polinom* , moștenind atributele și metodele acesteia din urmă. După cum sugerează și numele, această clasă modelează un alt caz particular de polinom, anume cel de gradul al doilea. Obiectele acestei clase sunt inițializate în cadrul constructorului cu același nume, care nu face decât să apeleze constructorul *super(int grad, int[] coef)* a clasei părinte, parametrii pentru acesta fiind determinați în funcție de parametrii primiți de constructorul curent. Gradul polinomului fiind doi, impune ca polinomul să poată avea cel mult două rădăcini reale. Întrucât metoda `solve(double startValue)` din clasa părinte nu poate returna decât o singură rădăcină, am ales supraîncărcarea acestei metode, astfel încât aceasta să returneze toate rădăcinile reale ale polinomului de grad doi.

Clasa *PolinomGrad3*, cea de a treia care reprezintă o clasă derivată a clasei *Polinom*, alături de celelalte două menționate mai sus, modelează polinomul de gradul al treilea. Obiectele sunt inițializate la fel, prin intermediul unui constructor cu același nume, în cadrul căruia se apelează constructorul clasei părinte, parametrii transmiși acestuia fiind determinați în funcție de parametrii primiți de constructorul clasei curente. Numărul maxim de rădăcini ale acestui polinom este dat de gradul polinomului, așadar trei. Din aceleași considerente ca cele amintite la polinomul de gradul al doilea, am ales supraîncărcarea metodei `solve()` , restul operațiilor implementându-se la fel ca în metodele moștenite din clasa părinte *Polinom.*

Clasa *InterfataUtilizator* reprezintă clasa care modelează interfața grafică cu utilizatorul (GUI) prin care un utilizator poate interacționa cu aplicația propriu-zisă. Ca bibliotecă de dezvoltare a interfeței grafice am folosit *JavaFX*. O aplicație JavaFX trebuie să extindă clasa *javafx.application.Application*. Aceasta este o clasă abstractă, deci suntem obligați să definim metoda *public void start(Stage primaryStage)*, care este metoda de start a aplicației.

Pentru folosirea aplicației trebuie adăgată librăria *JavaFX*. Pentru mai multe detalii urmăriți partea de început a tutorialului : https://www.youtube.com/watch?v=EEcpMTpaWhs.

3.2 Diagrama UML



3.3 Relațiile între clase

Pe baza diagramei UML, se observă relațiile existente între cele cinci clase : *Polinom, PolinomGrad1, PolinomGrad2, PolinomGrad3* și *InterfataUtilizator.*

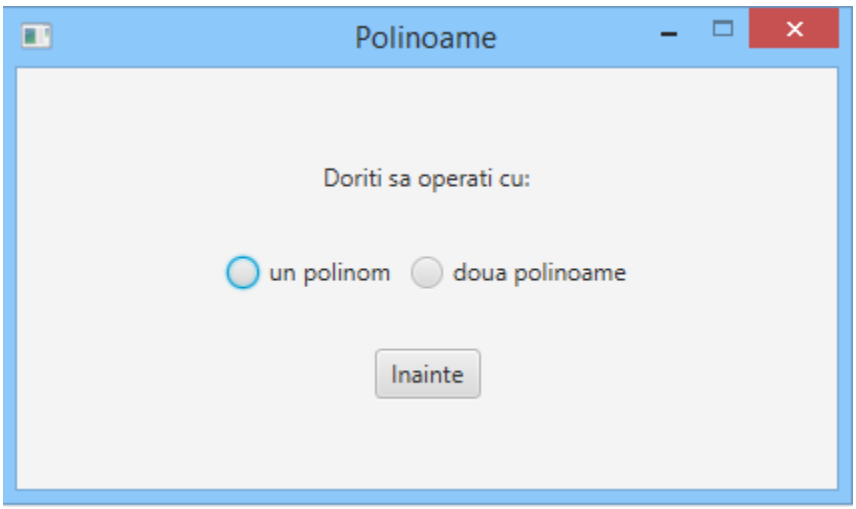
Între clasele *PolinomGrad1, PolinomGrad2, PolinomGrad3* și clasa *Polinom* se observă existența unei relații de generalizare, sau de moștenire simplă, conform căreia primele trei clase reprezintă clase derivate sau subclase ale clasei *Polinom*, definită în acest context clasă părinte sau superclasă. Se mai observă de asemenea că subclasele moștenesc metodele și atributele clasei părinte, și supraîncarcă metoda de determinare a rădăcinilor polinomului `solve()`.

Între clasa *InterfataUtilizator* și *Polinom* se observă că există o relație de asociere, care reprezintă cazul în care un obiect din clasa *InterfataUtilizator* folosește serviciile unui obiect al clasei *Polinom,* pentru a furniza utilizatorului rezultatele corespunzătoare la operațiile pe care acesta alege să le facă pe un sistem de polinoame.

3.4 Interfața utilizator

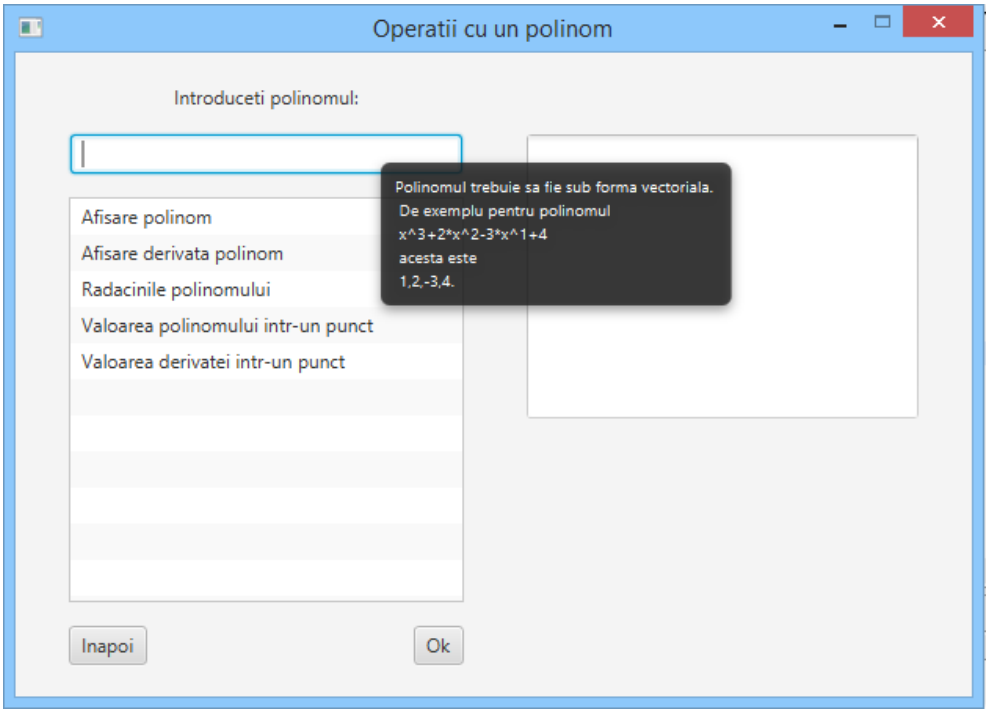
Așa cum am mai menționat, interfața grafică cu utilizatorul am implementat-o cu ajutorul bibliotecii JavaFX. Principalele clase folosite în cadrul realizării acestei interfețe sunt : *Button, RadioButton, Label, TextField, TextArea, ListView<T>, Vbox, Hbox etc.*

- Fereastra de început afișată la rularea aplicației este urmatoarea:



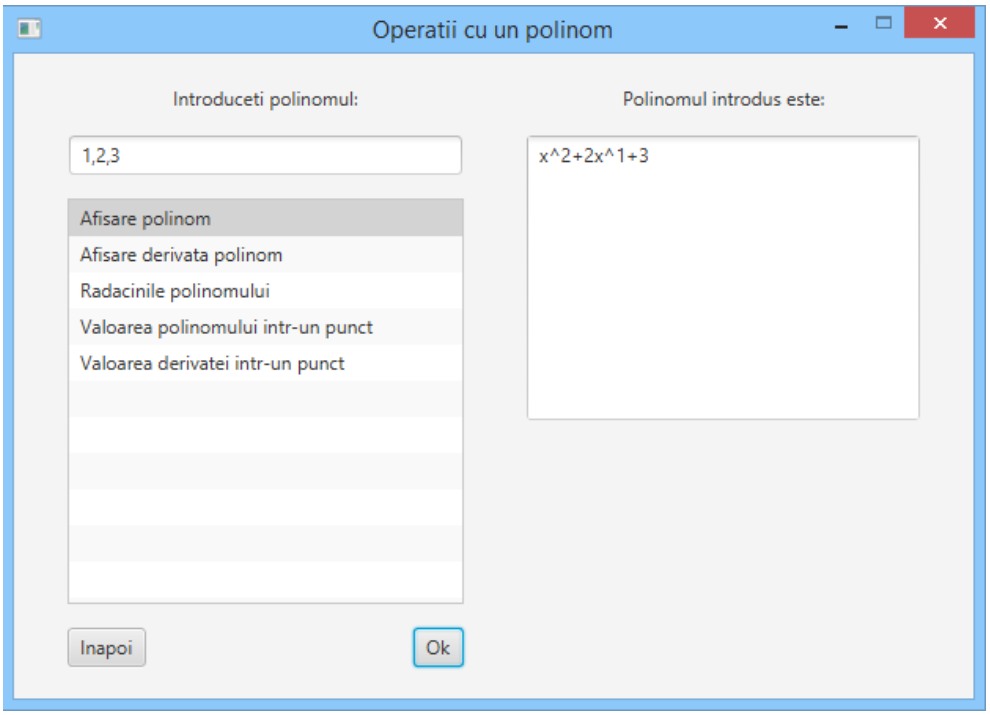
Utilizatorul trebuie să aleagă una dintre cele două opțiuni pentru a putea trece mai departe, respectiv prima opțiune dacă dorește să opereze cu un singur polinom, sau a doua opțiune dacă dorește să opereze cu un set de două polinoame. În cazul în care acesta dorește ulterior să aleagă cealaltă opțiune, va avea posibilitatea de a se întoarce la aceasta fereastră.

- În cazul în care utilizatorul a ales prima opțiune, el este redirecționat la următoarea fereastră:

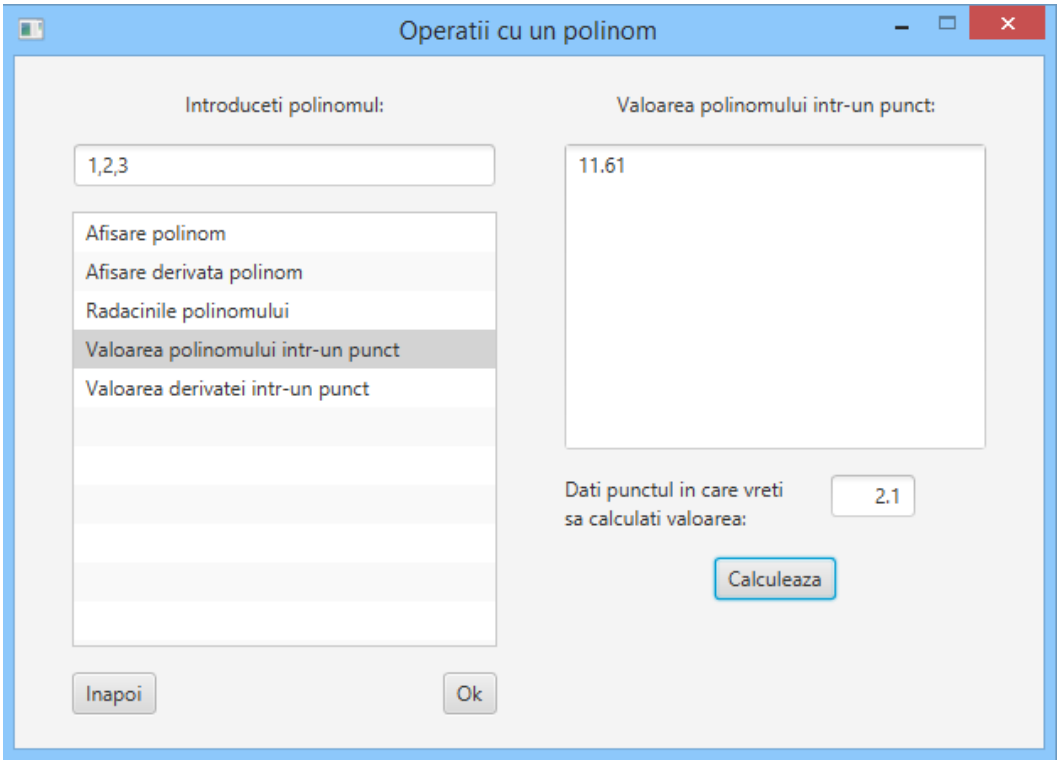


În partea din stânga putem observa un câmp de tip *TextField* în care utilizatorul trebuie să scrie polinomul pentru care vrea să efectueze operații. Așa cum am mai menționat, am ales ca introducerea polinomului de către utilizator să se facă sub formă vectorială, din acest motiv, atunci când utilizatorul poziționează mouse-ul în zona de introducere a polinomului, apare o căsuță, mai exact o componentă de tip *Tooltip* prin care este informat cum trebuie introdus acesta.

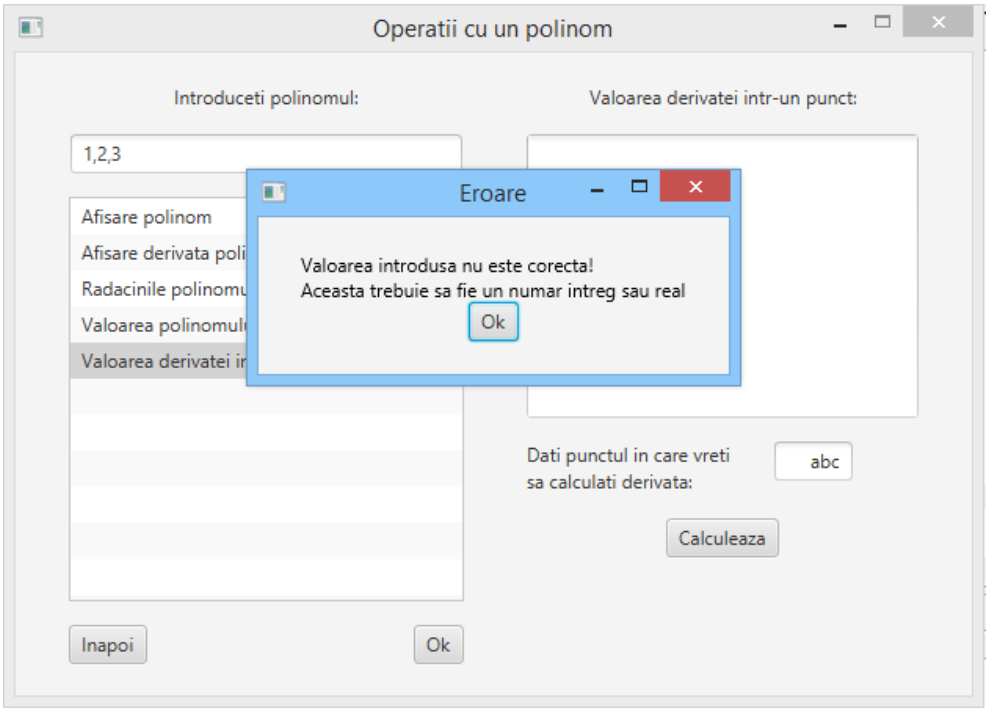
În continuare observăm un meniu care conține operațiile pe care utilizatorul poate să le folosească asupra polinomului tocmai introdus. În partea din stânga jos observăm două butoane, buton *Inapoi* care ne direcționează la fereastra anterioară și buton *Ok* care ne furnizează rezultatul operației alese asupra polinomului scris în partea din dreapta a ferestrei, în cadrul componentei de tip *TextArea*. Dacă, de exemplu, utilizatorul alege să vizualizeze polinomul introdus, fereastra va arăta astfel:



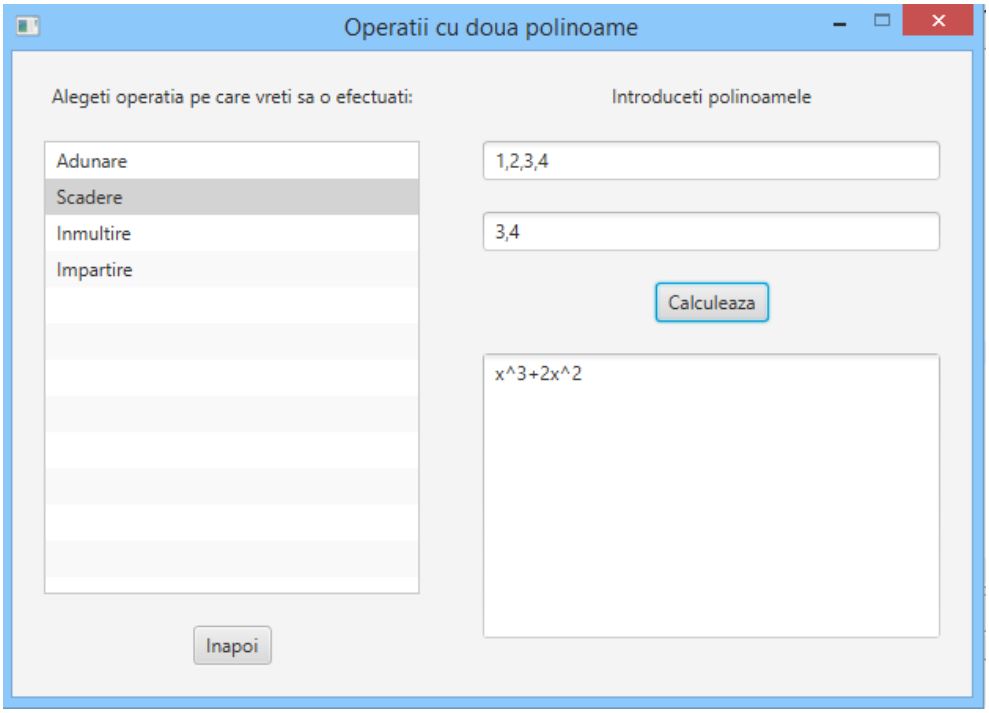
Dacă acesta va alege una dintre opțiunile de afișare a valorii polinomului într-un punct sau de afișare a valorii derivatei polinomului într-un punct, atunci în partea din dreapta vor apărea componente care să îi permită acestuia să introducă punctul în care dorește să calculeze valoarea:



Dacă, de exemplu, acesta nu introduce o valoare validă, el va fi atenționat printr-un mesaj de eroare de greșeala făcută:



Dacă utilizatorul a ales cea de-a doua opțiune din fereastra de start, acestuia i se va deschide o nouă fereastră:



Această fereastră este asemănătoare cu cea prezentată anterior, conține și ea o listă cu operațiile ce se pot efectua asupra polinoamelor, însă de data aceasta observăm două componente de tip *TextField* pentru introducerea a două polinoame. În partea din stânga jos observăm același buton *Inapoi* ce trimite utilizatorul la fereastra anterioară, iar în partea dreaptă un alt buton *Calculeaza*, care odată apăsat, va genera afișarea rezultatului operației alese între cele două polinoame.

3.5 Algoritmi

Algoritmii utilizați la implementarea operațiilor sunt :

- Pentru operația de adunare am comparat mai întâi gradul celor două polinoame care trebuiesc adunate. În cazul ideal, când ambele polinoame au același grad, se parcurg coeficienții polinoamelor și se adună, rezultatul fiind păstrat într-un tablou auxiliar. În acest caz trebuie să avem grijă ca polinomul rezultat să nu aibă primul coeficient 0, iar pentru aceasta translatăm vectorul de coeficienți la stânga atâtâ timp cât pe prima poziție întalnim valoarea 0. Dacă gradele celor două polinoame diferă, atunci pe pozițiile tabloul rezultat până la gradul celui mai mic polinom punem coeficienții polinomului de grad mai mare, după care punem suma coeficienților corespunzători celor două polinoame.
- Pentru operația de scădere trebuie să verificăm condiția că gradul primului polinom (descăzutul) este mai mare sau egal cu gradul celui de-al doilea polinom (scăzătorul). Algoritmul de scădere este asemănător cu cel de la adunare.
- Pentru operația de înmulțire gradul polinomului rezultat este egal cu suma dintre gradele polinoamelor care se înmulțesc. Fiecare coeficient este dat de produsul coeficienților de același grad adunat la vechiul coeficient de pe acea poziție.
- Pentru operația de împărțire gradul câtului este dat de diferența dintre gradul deîmpărțitului și gradul împărțitorului. Algoritmul în pseudocod este următorul:

```
function n / d:
  require d ≠ 0
  (q, r) ← (0, n)             # At each step n = d × q + r
  while r ≠ 0 AND degree(r) ≥ degree(d):
    t ← lead(r)/lead(d)      # Divide the leading terms
     (q, r) ← (q + t, r - (t * d))
  return (q, r)
```

4.Implementare și testare

4.1 Implementare

 Clasa *Polinom* modelează conceptul de polinom general. Metodele acestei clase sunt:

- `getCoef()` - returnează coeficienții polinomului sub formă de tablou de întregi
- `getGrad()` - returnează gradul polinomului
- `setCoef(int[] coefNoi)` – setează coeficienții polinomului cu coeficienții transmiși ca parametru
- `setGrad(int gradNou)` – setează gradul polinomului la gradul transmis ca parametru
- `calculeazaDerivata()` – returnează un obiect de tip Polinom, reprezentând derivata polinomului
- `toString()` – returnează un șir de caractere de forma *„x^3+2x^2-5x^1-100”* obținut în funcție de gradul și coeficienții polinomului pentru care se apelează metoda
- `getFuncValue(double x)` – returnează o valoare reală reprezentând valoarea polinomului pentru care se apelează metoda în punctul x transmis ca parametru
- `getDerivValue(double x)` – returnează o valoare reală reprezentând valoarea derivatei polinomului pentru care se apelează metoda în punctul x transmis ca parametru ; această metodă apelează la rândul ei metodele `calculeazaDerivata()` și `getFuncValue(double x)` pentru calcularea mai ușoară a rezultatului
- `add(Polinom p)` – returnează un obiect de tip Polinom, reprezentând suma dintre polinomul pentru care se apelează metoda și polinomul transmis ca parametru
- `substract(Polinom p)` – returnează un obiect de tip Polinom, reprezentând diferența dintre polinomul pentru care se apelează metoda și polinomul transmis ca parametru ; gradul primului polinom trebuie să fie mai mare sau egal decât gradul celui de-al doilea polinom pentru ca scăderea celor două să poată fi efectuată

- `multiply(Polinom p)` – returnează un obiect de tip Polinom, reprezentând produsul dintre polinomul pentru care se apelează metoda și polinomul transmis ca parametru ; gradul polinomului rezultat este egal cu suma gradelor polinoamelor între care are loc operația

- `divide(Polinom p)` – returnează un șir de caractere ce conține atât reprezentarea câtului cât și a restului împărțirii polinomului pentru care se apelează metoda și polinomul transmis ca parametru ; am ales ca această metodă să nu returneze un set de polinoame(câtul și restul) pentru că polinomul rezultat ar putea avea coeficienți reali, iar clasa Polinom este modelată ca având coeficienții întregi. Un alt motiv este acela că în cadrul aplicației nu este nevoie decât de reprezentarea rezultatului sub formă de șir de caractere de forma *„x^3+2.5x^2-x^1+2.12”*.

- `solve(double startValue)` – returnează un tablou de valori reale reprezentând rădăcinile polinomului ; implementarea metodei se bazează pe *Metoda lui Newton*, care oferă o aproximare a rădăcinii în funcție de un punct de pornire transmis ca parametru.

Clasa *PolinomGrad1* modelează o particularitate a polinoamelor, respectiv polinomul de grad întâi. Acesta extinde clasa *Polinom*, moștenind atributele și metodele acesteia. Metoda supraîncărcată de această clasă este metoda :

- `solve()` – returnează unica rădăcină reală a polinomui și anume $x = -\frac{b}{a}$

Clasa *PolinomGrad2* modelează o particularitate a polinoamelor, respectiv polinomul de grad al doilea. Acesta extinde clasa *Polinom*, moștenind atributele și metodele acesteia. Metoda supraîncărcată de această clasă este metoda :

- `solve()` – returnează cel mult două rădăcini reale, după cum este explicat în secțiunea 2.II.

Clasa *PolinomGrad3* modelează o particularitate a polinoamelor, respectiv polinomul de grad al treilea. Acesta extinde clasa *Polinom*, moștenind atributele și metodele acesteia. Metoda supraîncărcată de această clasă este metoda :

- `solve()` – returnează cel mult trei rădăcini reale, după cum este explicat în secțiunea 2.III.

Clasa *InterfataUtilizator* reprezintă clasa care modelează interfața grafică cu utilizatorul (GUI) prin care un utilizator poate interacționa cu aplicația propriu-zisă. Metodele acestei clase sunt :

- `start(Stage stage)` – reprezintă metoda de start a aplicației. În cadrul acestei metode este setat pentru început atributul *primaryStage* cu un obiect de tip *Stage* primit ca parametru, pentru a ști ulterior să ne reîntoarcem la fereastra principală. În această metodă este asociată declanșarea apăsării unui buton numit *Inainte* cu o metodă de tip *handler*. Această metodă duce la apelarea metodelor `fereastraOperatiiPolinom()` sau `fereastraOperatiiPolinoame()` în funcție de ce buton radio este ales de utilizator în momentul apăsării butonului *Inainte.*

- `fereastraOperatiiPolinom()` – reprezintă o metoda ce duce la apariția unei ferestre din care utilizatorul poate efectua anumite operații asupra unui polinom. Și în această metodă au loc asocieri între declanșări de evenimente și metode de tip *handler*. La apăsarea butonului *Ok* și/sau *Calculeaza* sunt generate rezultate în funcție de datele introduse de utilizator și operațiile alese, iar la apăsarea butonului *Inapoi* se revine la fereastra de start, respectiv se „ascunde” fereastra curentă și se apelează metoda `start(primaryStage)`

- `fereastraOperatiiPolinoame()` – reprezintă o metoda ce duce la apariția unei ferestre din care utilizatorul poate efectua anumite operații asupra unui set de două polinoame. Și în această metodă au loc asocieri între declanșări de evenimente și metode de tip *handler*. La apăsarea butonului *Calculeaza* sunt generate rezultate în funcție de datele introduse de utilizator și operația aleasă, iar la apăsarea butonului *Inapoi* se revine la fereastra de start, respectiv se „ascunde” fereastra curentă și se apelează metoda `start(primaryStage)`

- `afiseazaMesajEroare(String s)` – reprezintă o metodă apelată în cazul apariției unor excepții care afișează o fereastră ce conține un mesaj de eroare, transmis ca parametru.

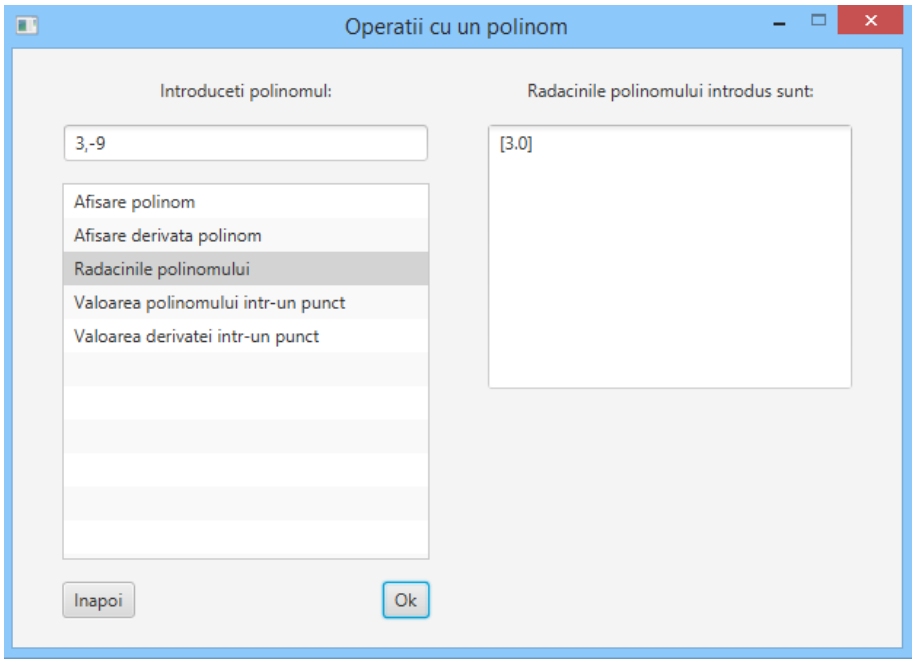
- `getCoef(String sir)` – reprezintă o metodă care returnează un tablou de întregi reprezentând coeficienții polinomului din șirul introdus de utilizator. De exemplu, șirul introdus sub forma „1,2,3,4” de către utilizator este „spart” cu ajutorul unui obiect de tip *StringTokenizer* într-un tablou de forma [1,2,3,4].

4.2 Testare

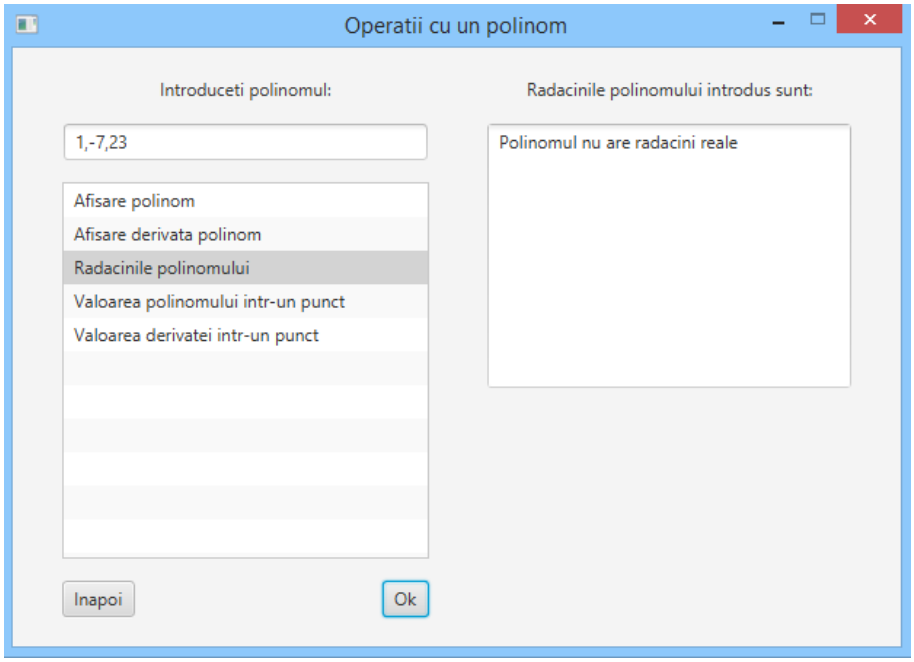
Testarea operațiilor implementate în clasa *Polinom* s-a realizat folosind *TestNG*, o platformă de testare bazată pe adnotări. Această facilitate permite posibilitatea testării codului pe parcursul evoluției programului. Testele făcute se pot refolosi, chiar dacă sunt aduse modificări la metoda pe care o testează.

Pentru testarea aplicației finale vom prezenta câteva cazuri de testare, pentru a verifica dacă aceasta funcționează corect :

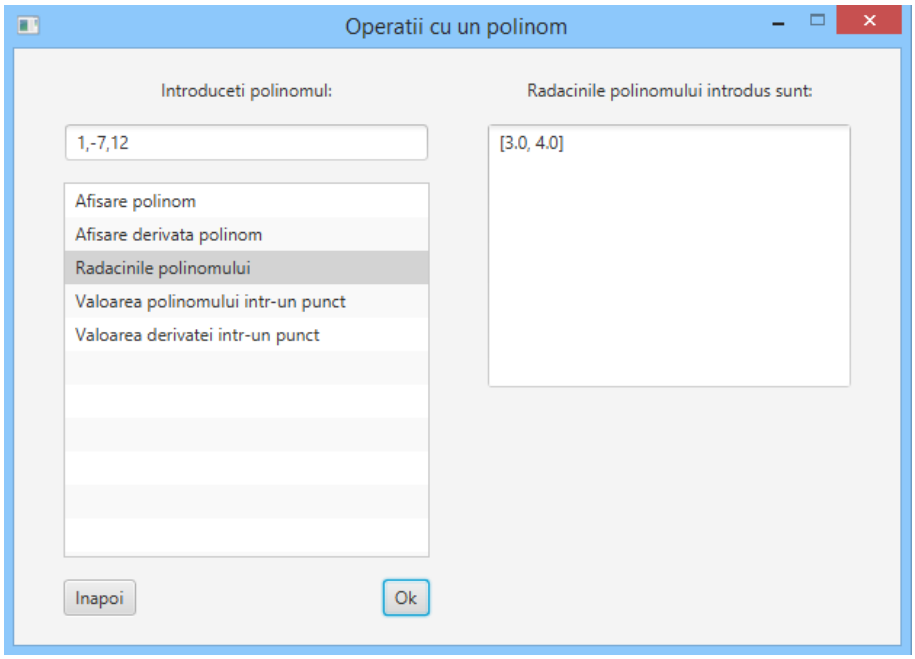
- ✓ Pentru operația de găsire a rădăcinilor, introducem date de intrare corespunzătoare fiecărui caz particular de polinom, pentru a vedea dacă rădăcinile furnizate sunt corecte.
  - Polinom de gradul întâi : *3x-9*



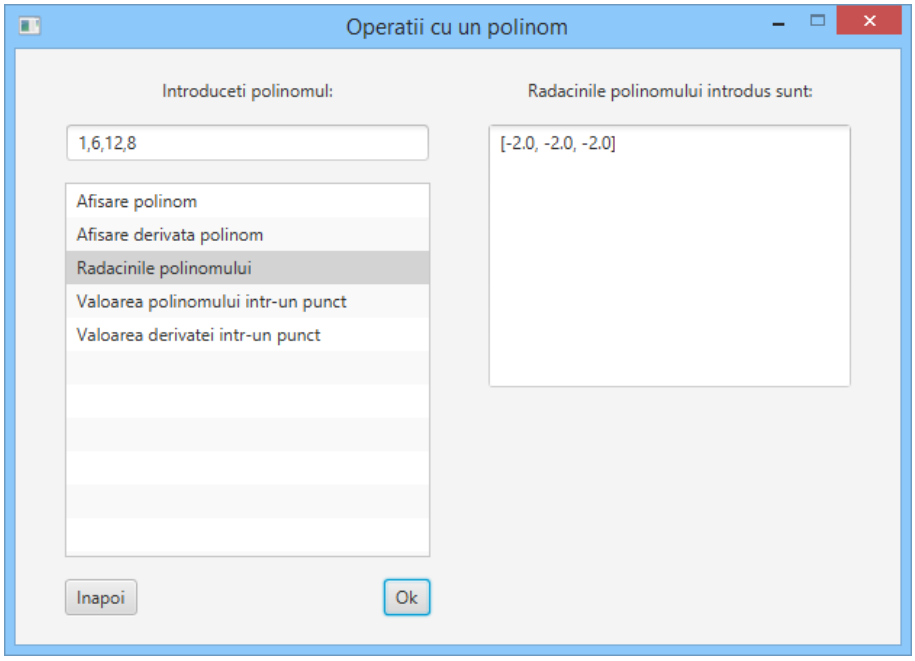
  - Polinom de gradul al doilea : *x^2-7x+23 (nu are rădăcini reale)*



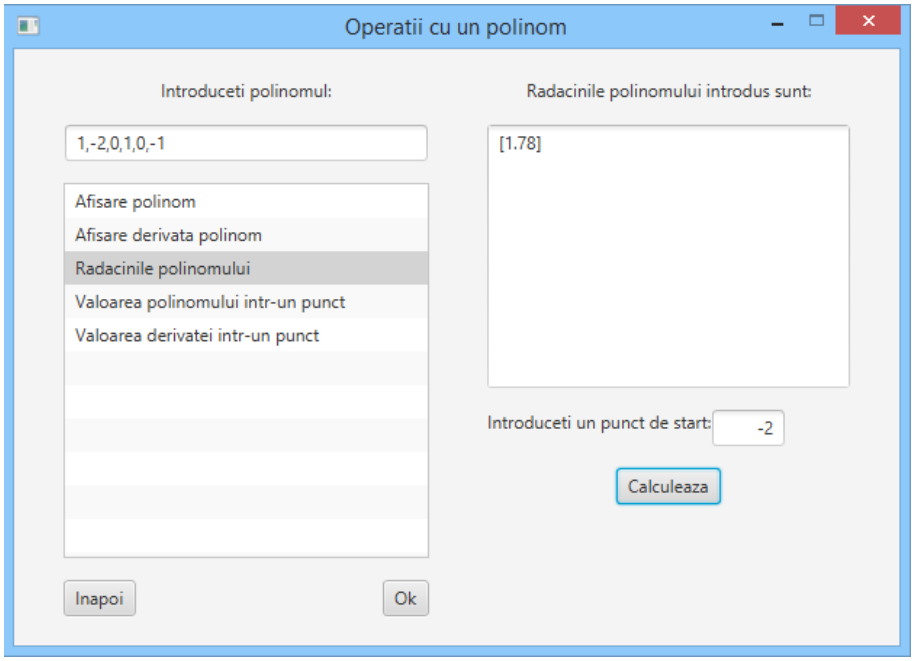
- Polinom de gradul al doilea : *x^2-7x+12(are rădăcinile 3 și 4)*



- Polinom de gradul al treilea: *x^3+6x^2+12x^1+8 (are toate rădăcinile egale cu -2)*

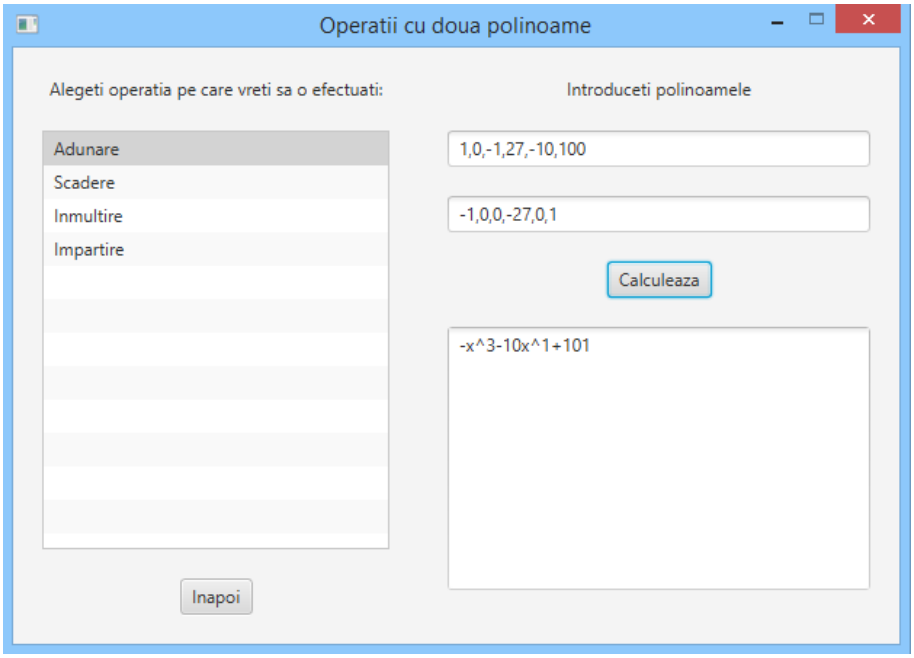


- Polinom de grad cinci *x^5-2x^4+x^2-1* cu punctul de start *x=-2*



✓ Pentru operațiile pe două polinoame, introducem date valide și/sau invalide, pentru a vedea ce rezultate ne oferă sistemul:

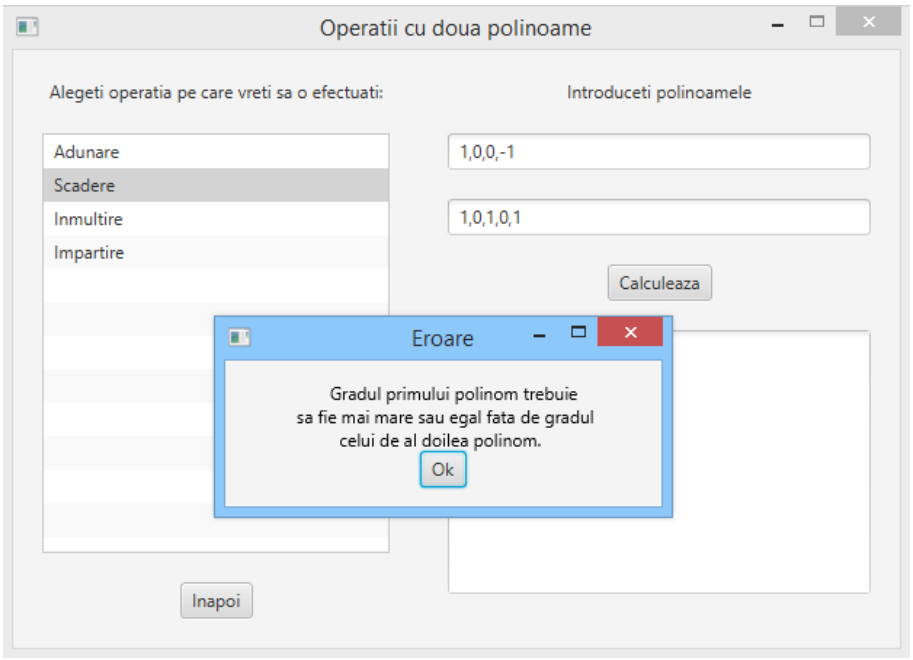
- Adunarea polinoamelor *x^5-x^3+27x^2-10x^1+100 și –x^5-27x^2+1*

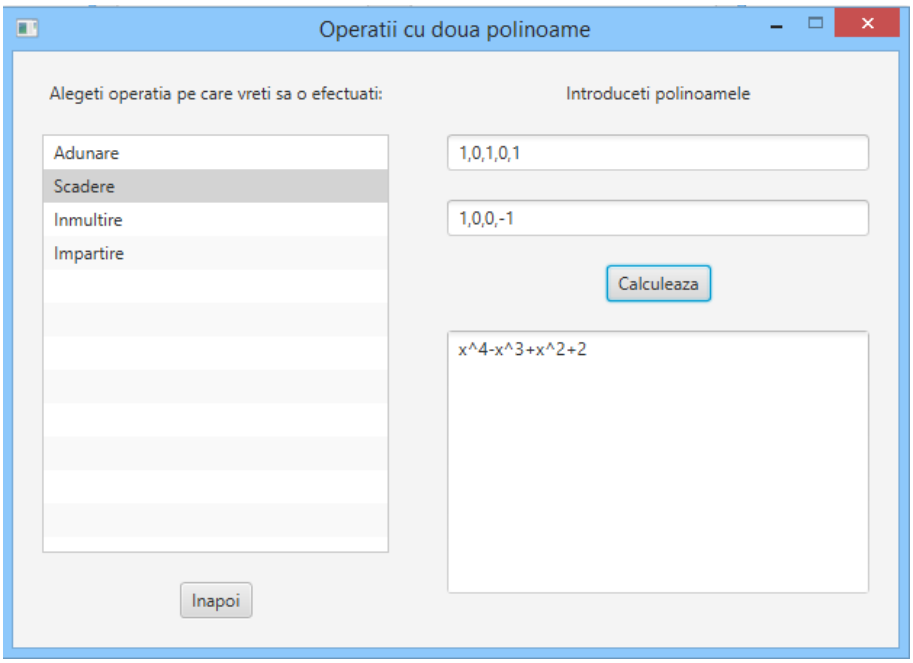


- Scăderea polinoamelor

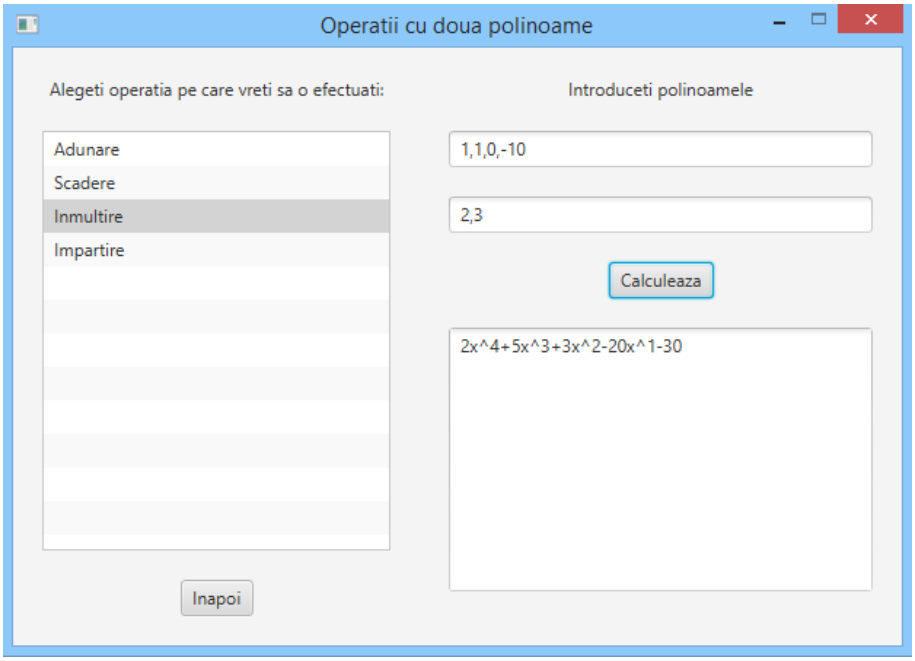
  x^3-1 și *x^4+x^2+1 (scăderea nu poate avea loc in acest caz)*

  x^4+x^2+1 și x^3-1

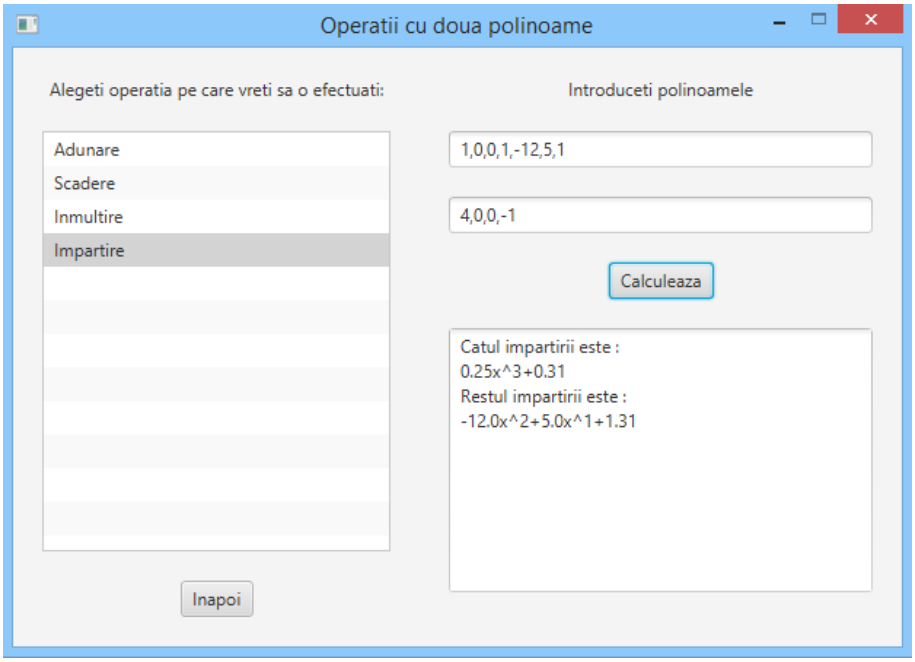




- Înmulțirea polinoamelor : *x^3+x^2-10 și 2x^1+3*



- Împărțirea polinoamelor : *x^6+x^3-12x^2+5x^1+1 și 4x^3-1*



5. Rezultate

În urma testării aplicației, se observă că aceasta rulează fără probleme indiferent de gradul polinomului introdus sau de operațiile alese. Se mai observă că atunci când utilizatorul introduce date de intrare invalide acesta este atenționat cu mesaje de eroare corespunzătoare.

6. Concluzii și dezvoltări ulterioare

În concluzie, se poate spune că am reușit să realizez un sistem de procesare a polinoamelor, care reușește să implementeze operațiile principale asupra unui polinom sau a unui set de polinoame. Deși implementarea nu a ridicat probleme mari, am întâmpinat dificultăți la crearea interfeței grafice. Cu toate acestea, am reușit să învăț multe lucruri despre cum se crează o interfață grafică, în special despre biblioteca de dezvoltare JavaFX.

Chiar dacă aplicația respectă cerințele problemei, i se pot aduce îmbunătățiri precum:

- creare unei clase care modelează un polinom cu coeficienți reali, pentru a nu mai întâmpina probleme la operația de împărțire
- adăugarea unor operații pentru polinoame precum : integrare fără limite de integrare, integrare cu limite de intregrare, afișarea graficului unei funcții etc.

- modificarea interfeței grafice
- modificarea metodei `getCoef(String sir)` astfel încât aceasta să extragă coeficienții unui polinom dacă el este introdus de utilizator sub forma „ax^n+bx^(n-1)+....”

7. Bibliografie

http://www.1728.org/cubic2.htm

http://en.wikipedia.org/wiki/Polynomial_long_division

http://docs.oracle.com/javase/8/javafx/api/toc.htm