

**Documentatie Tema 1**

**Calculator de Polinoame**

**Codrescu Razvan-Gabriel**

**Grupa 30224**

**Profesor Laborator Antal Marcel**

**Cuprins**

**1.Obiectivul temei**

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

**3. Proiectare (diagrame UML, structure de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)**

**4. Implementare**

**5. Cazuri de testare**

**6.Use-case**

**7. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare 8.Bibliografie**

**1.Obiectivul temei**

*Enunt:* Propuneti, proiectati si implementati un sistem de procesare a polinoamelor de o singura variabila cu coeficienti reali.

Aceasta tema are ca obiectiv crearea unui calculator de polinoame. La fel ca si calculatorul de buzunar, calculatorul sistemului de operare pe care il folositi sau orice alt tip de calculator existent, acesta trebuie sa implementeze operatii precum, adunarea, scaderea, inmultirea, impartirea a doua polinoame, precum si derivarea unuia si integrarea lui. La fel ca si orice sistem de calcul si acesta are date de intrare si data de iesire date de rezultatul calculelor.

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

Principiile de baza sunt:

**1.Abstractizare** = Este posibilitatea ca un program să separe unele aspecte ale informației pe care o manipulează, adică posibilitatea de a se concentra asupra esențialului. Fiecare obiect în sistem are rolul unui “actor” abstract, care poate executa acțiuni, își poate modifica și comunica starea și poate comunica cu alte obiecte din sistem fără a dezvălui cum au fost implementate acele facilitați.Are loc dupa analiza problemei si specificarea cerintelor.

**2.Încapsularea** = numită și ascunderea de informații: Asigură faptul că obiectele nu pot schimba starea internă a altor obiecte în mod direct (ci doar prin metode puse la dispoziție de obiectul respectiv); doar metodele proprii ale obiectului pot accesa starea acestuia. Fiecare tip de obiect expune o interfață pentru celelalte obiecte care specifică modul cum acele obiecte pot interacționa cu el.

**3.Polimorfismul** = Este abilitatea de a procesa obiectele în mod diferit, în funcție de tipul sau de clasa lor. Mai exact, este abilitatea de a redefini metode pentru clasele derivate. De exemplu pentru o clasă Figura putem defini o metodă arie. Dacă Cerc, Dreptunghi, etc. ce vor extinde clasa Figura, acestea pot redefini metoda arie.

**4.Moștenirea** = Organizează și facilitează polimorfismul și încapsularea, permițând definirea și crearea unor clase specializate plecând de la clase (generale) deja definite - acestea pot împărtăși (și extinde) comportamentul lor, fără a fi nevoie de a-l redefini. Aceasta se face de obicei prin gruparea obiectelor în clase și prin definirea de clase ca extinderi ale unor clase existente. Conceptul de moștenire permite construirea unor clase noi, care păstrează caracteristicile și comportarea, deci datele și funcțiile membru, de la una sau mai multe clase definite anterior, numite clase de bază, fiind posibilă redefinirea sau adăugarea unor date și funcții noi.

Calculator ce urmeaza sa fie implementat va avea coeficienti reali. Au fost implementate urmatoarele operatii: adunare, scadere, inmultire, impartire, derivare si integrare.

Pentru a putea implementa problema, trebuie sa vedem datele de care avem nevoie si sa vedem cum vrem sa le transmitem ca parametri in functii si cum le conferim o structura cat mai usor de manipulat. De asemenea, trebuie creata o interfata cat mai practica si usor de utilizat.

Pentru realizarea operatiilor intre polinoame vom aveam nevoie sa stim coeficientul si puterea fiecarui membru. Pentru a fi cat mai usor si pentru utilizator de introdus si de citit, polinoamele se vor da sub forma de string – “2x^2+3x^3+1x^0”. Pentru rezolvarea acestei probleme trebuie sa luam in considerare procesarea polinoamelor(parsarea stringului), modelarea fiecarei operatii si afisarea rezultatului dorit.

**3. Proiectare (diagrame UML, structure de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)**

Am implementat urmatoarele clase pentru rezolvarea cerintei: Monom, Polinom, Operatii, ResDiv, TestOperatii, Frame, Controller, UniversalFrame. Toate functiile au variabilele declarate private, acestea fiind accesate cu gettere si settere. Programul este realizat dupa conventia MVC(Model, View, Controller). View se ocupa de cum sa arate interfata programului(Frame), Cotroller se ocupa de fiecare efectuare de actiuni pe suprafata interfetei(ActionListener) si ce operatii sa se execute la apasarea acestora iar Model contine codul propriu-zis al programului(ce se ascunde in spatele interfetei).

Clasa **Monom** are ca parametri puterea si coeficientul monomului. Exista metoda Monom care primeste un String, si il separa utilizand functia split care “sparge” sirul introdus ca parametru dupa un regex introdus si il pune intr-un sir de String-uri. Aceasta arunca exceptie in caz ca citirea nu a fost efectuata corect, exceptia urmand sa fie prinsa in codul fiecarei operatii din Controller. De asemenea, mai este implementata metoda compareTo care va fi folosita in Polinom(Collections.sort()) pentru a sorta monoamele in ordinea crescatoare a puterilor.

Clasa **Polinom** prezinta ca si parametru un ArrayList de Monoame si gradul maxim al polinomului. In clasa respective exista metoda Polinom care primeste un String si il converteste intr-o lista de Monoame cu ajutorul functiei split. La sfarsit, polinomul este sortat si gradul maxim este salvat in gradMax. Metoda searchGradMax este utilizata pentru a gasi gradul maxim in cazul unei modificari suferite de catre polinom(spre exemplu in metoda de impartire a celor 2 polinoame). De asemenea a fost implementata metoda toString pentru afisarea corespunzatoare a polinoamelor.

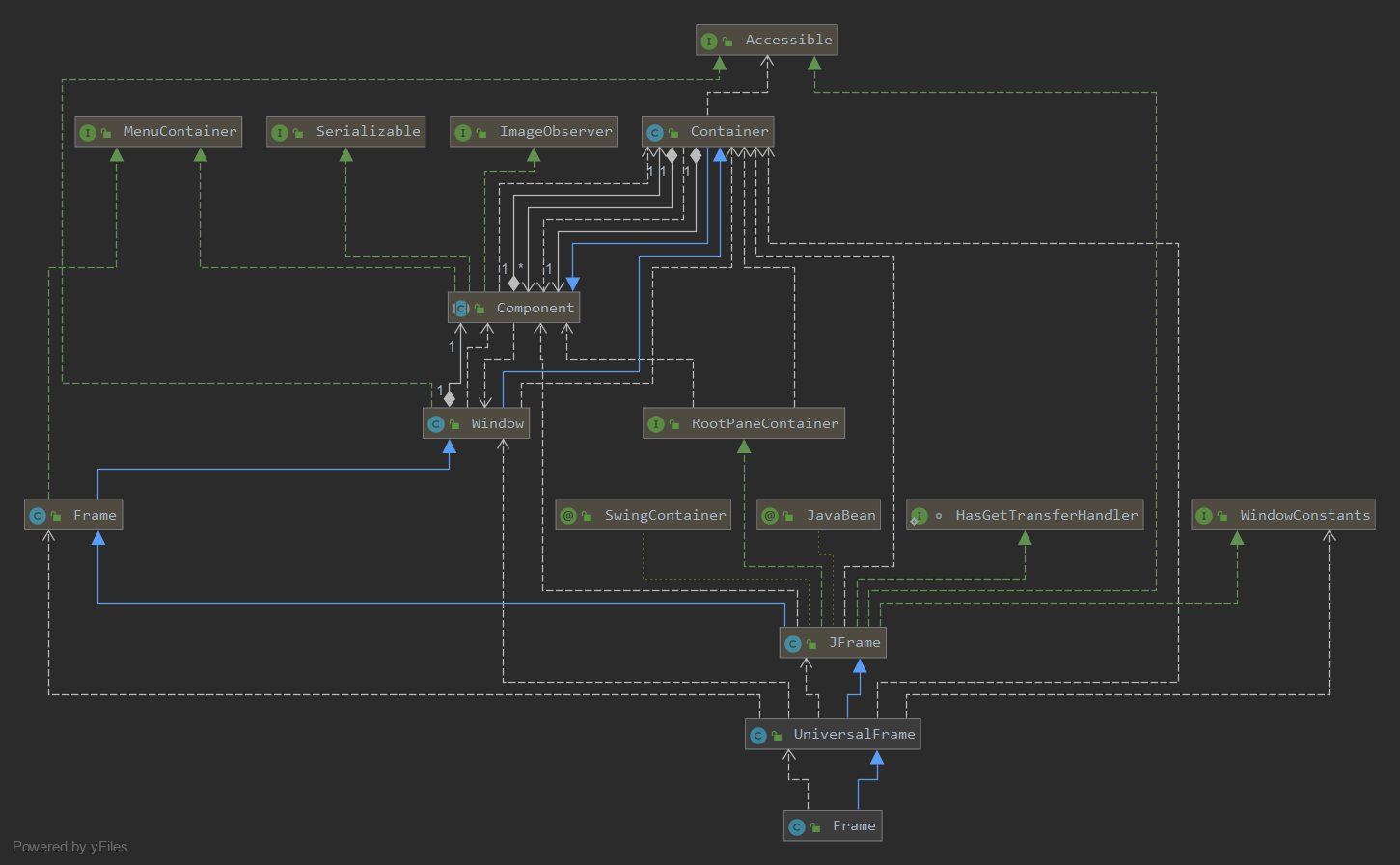
Clasa **Operatii** implementeaza cele 6 operatii cerute: Adunare, Scadere, Inmultire, Impartire, Derivare si Integrare. S-a efectuat in operatii crearea de polinoame noi pentru a manipula mai usor informatii de pe cele introduse.

**UniversalFrame** este clasa unde se stabileste dimensiunea frame-ului, fontul scrisului, culoarea background-ului ferestrei, si titlul ferestrei. EXIT\_ON\_CLOSE face ca la inchiderea ferestrei, programul sa fie si el oprit.

In **Controller** este implementat **ActionListener**-ul fiecarui buton, plus la fiecare ActionListener este cate un try{}catch care sa anunte in caz ca polinomul nu a fost introdus sau a fost introdus gresit.

In **Frame** este stabilita pozitia in contentPanel a fiecarui buton, label, textField si checkBox si culoarea, fontul si dimensiunea acestora. Fiecare set de instructiuni a fost separate printr-un mic comentariu cu sectiunea ce urmeaza sa fie prezentata in cod. Dupa realizarea acestora, au fost implementate gettere care au fost declarate statice pentru fiecare piesa existenta a acestui program.

**Diagrama UML si dependentele**



**4.Implemetare**

**public static Polinom add(Polinom p1, Polinom p2)**

Metoda add primeste doua polinoame ca si parametri. Listele celor 2 polinoame vor fi adaugate intr-o singura lista care va fi sortata in ordine crescatoare a puterilor. Cat timp lista respectiva nu este goala, se vor aduna toti coeficientii polinoamelor cu aceeasi putere(monoamele cu puteri egale vor fi consecutive asezate in noul polinom datorita sortarii efectuate inainte) si se va adauga intr-o lista noua doar rezultatul lor, iar daca puterile monoamelor sunt diferite, se va adauga doar primul monom la lista noua. In final, se va return polinomul cu noua lista ca parametru.

**Public static Polinom sub(Polinom p1, Polinom p2)**

Scaderea(sub) este realizata asemanator, diferenta fiind ca cele doua polinoame sunt parcurse cu iteratori: cat timp nu s-a ajuns la finalul polinomului 2, se va parcurge polinomul 1(de asemenea cu ajutorul unui iterator) si se va scadea elementul in functie de daca se gaseste sau nu un element cu aceeasi putere, si se va salva in polinomul 1. Pentru verificarea gasirii si efectuarii operatiilor de scadere si adaugare in polinomul nou, am luat o variabila ok de timp int, care sa verifice daca s-au gasit elemente cu aceeasi putere. In ca contrar, elementul din polinomul doi va fi scazut si salvat in polinomul 1.

**public static Polinom mul(Polinom p1, Polinom p2)**

Inmultirea polinoamelor va fi realizata prin parcurgerea polinoamelor si inmultirea coeficientilor respectiv adunarea puterilor iar monomul rezultat din fiecare va fi adaugat intr-o lista noua. Lista noua va fi sortata si va fi parcursa, in caz ca sunt elemente cu aceeasi putere, acestea sa fie adunate intre ele, dupa algoritmul utilizat si la adunare: cat timp lista de monoame nu este goala, se verifica daca doua elemente consecutive au aceeasi putere si in caz afirmativ, se aduna, se adauga in lista noua si se elimina din cea veche. Altfel, acestea pur si simplu vor fi adaugate in noua lista si eliminate din cea veche.

**public static Polinom div(Polinom p1, Polinom p2)**

Impartirea este realizata dupa algoritmul prezentat la laborator. A fost necesar crearea unei noi clase – **ResDiv** – unde au fost salvate cele doua polinoame de cat si rest.Mai intai se vor face 2 verificari : daca cel de-al doilea polinom este 0, atunci rezultatul va fi nul; daca cel de-al doilea polinom are gradul 1, atunci se va efectua o impartire simpla a fiecarui coeficient din lista de monoame. Cat timp gradul maxim al primului polinom este mai mare decat gradul maxim al celui de-al doilea, se va efectua impartirea primelor doua monoame cu ajutorul unei metode div din clasa Monom, care primeste ca paramtri 2 monoame si va return monomul rezultat in urma impartirii celor 2, dupa care se va face suma dintre rezultatul actual si rezultatul impartirii precedente. Va fi creat un polinom auxiliar care sa ia valoarea rezultata din inmultirea polinomului 2 cu rezultatul impartii celor doua monoame(acesta reprezentand o amplificare maxima a polinomului 2), acesta urmand sa fie scazut din polinomul 1(realizand astfel restul impartirii).

**public static Polinom derivate(Polinom p1)**

Derivata va fi calculate prin parcurgerea polinomului si inmultirea coeficientului fiecarui monom cu puterea actuala si decrementarea puterii. Rezultatul va fi salvat in polinomul introdus ca parametru si returnat.

**public static Polinom integrate(Polinom p1)**

Integrata este realizata ca si derivata: polinomul este parcurs si puterea fiecarui monom este incrementata de aceasta data si coeficientul este impartit la puterea actuala, rezultata in urma incrementarii. Rezultatul este salvat in polinomul initial si returnat.

In **TestOperatii** s-a folosit JUnit pentru a testa cu ajutorul unor asertiuni fiecare dintre cele 6 operatii prezentate mai sus pe 2 polinoame exemplu. Asertiunile sunt de tip assertEquals() care primeste ca parametri 2 stringuri si le compara. In cazul in care sunt identice, testul va fi realizat cu success.

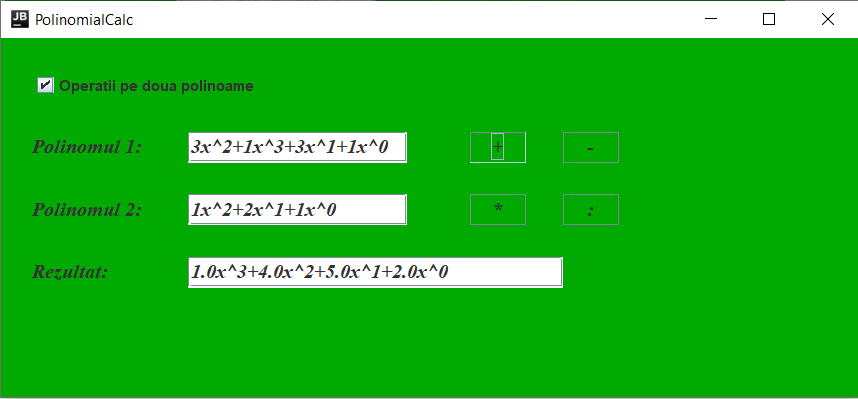
**5.Cazuri de testare**

Pentru fiecare caz vom lua urmatoarele doua polinoame pentru testare :

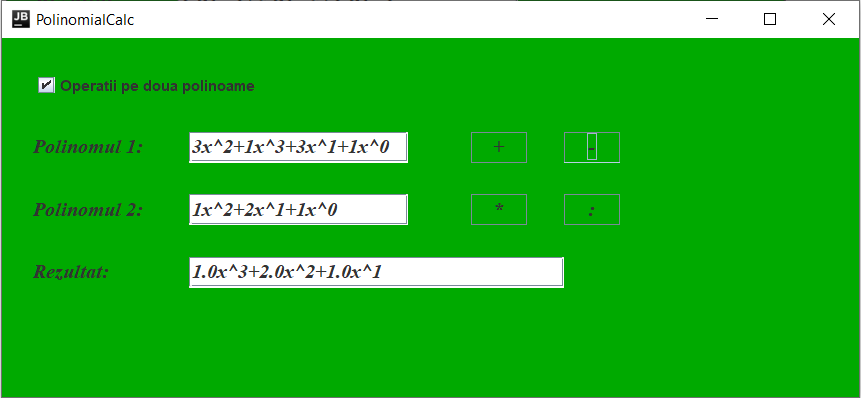
Pol1 = “3x^2+1x^3+3x^1+1x^0”

Pol2 = “1x^2+2x^1+1x^0”

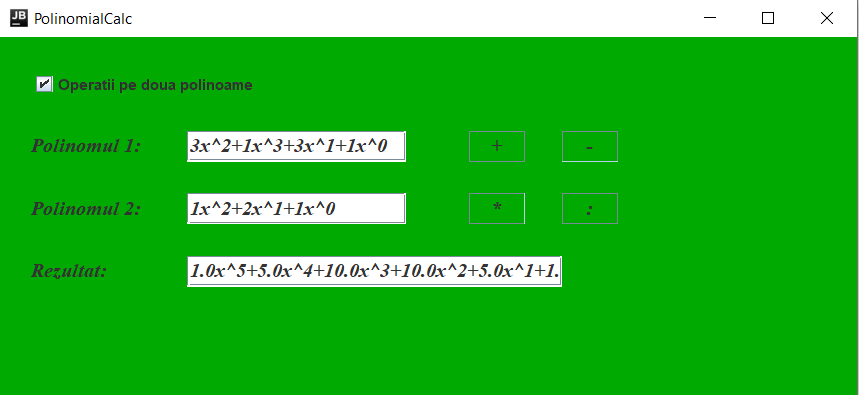
1.Adunarea



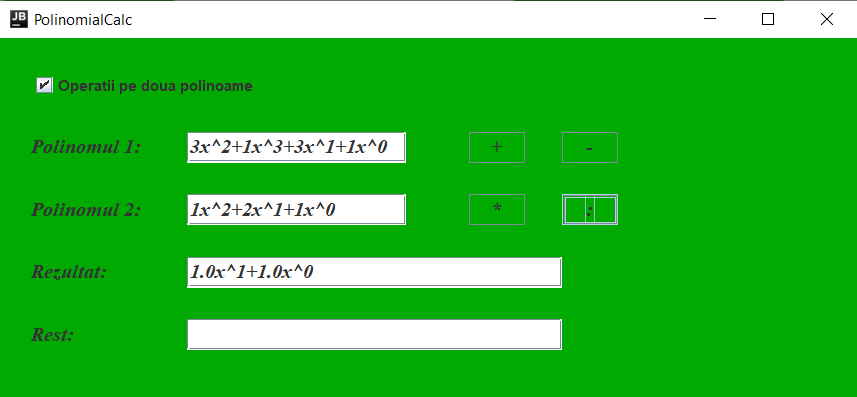
2.Scaderea



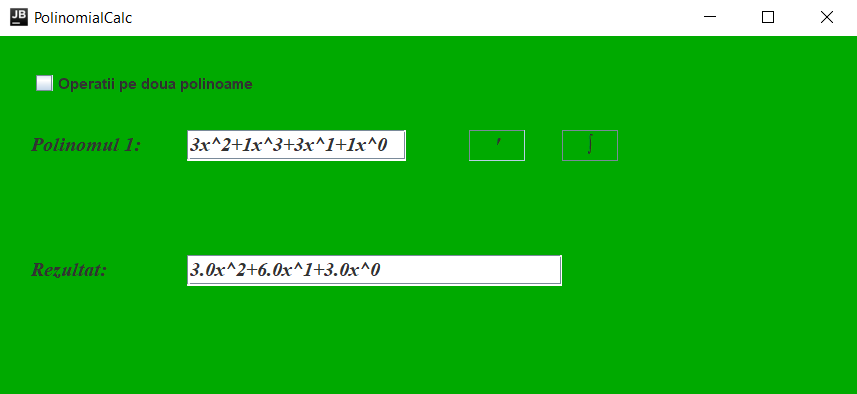
3.Inmultirea



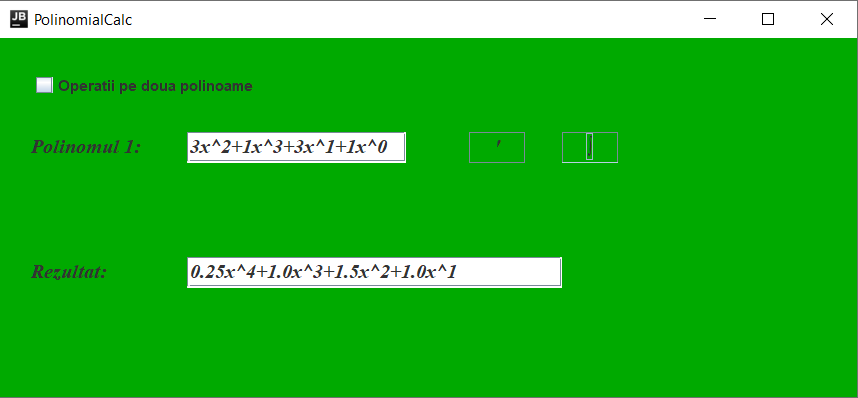
4.Impartirea(campul “Rest” este gol deoarece in urma impartirii celor 2 polinoame, restul este 0)



5.Derivarea

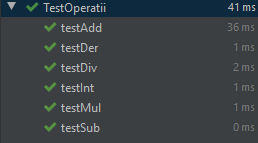


6.Integrarea



7.Teste

Se poate observa ca pentru fiecare operatie efectuata pe aceste polinoame, testele au fost efectuate cu success.



**6.Use-case**

In urmatoarele randuri, voi prezenta ce pasi trebuie sa parcurca utilizatorul pentru a efectua operatiile dorite cu success.

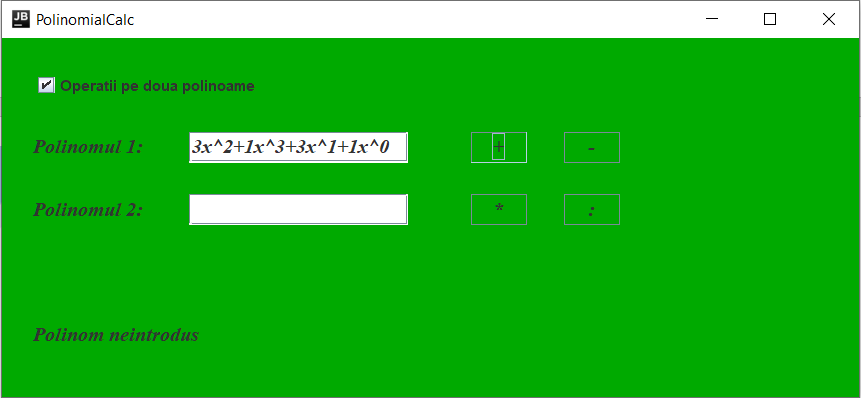
Pas1: Utilizatorul va porni aplicatia.

Pas2: Utilizatorul, in functie de ce fel de operatii vrea sa efectueze, va selecta/deselecta checkBox-ul pentru a ramane numarul de campuri necesare in care trebuie introduse polinoame pentru a efectua operatia dorita.

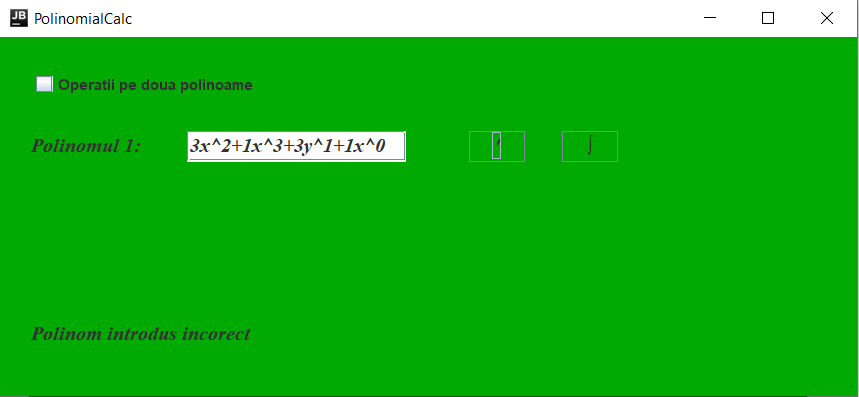
Pas3: Utilizatorul va scrie in campul/ campurile destinate polinoamelor(campul din dreapta textelor “Polinom 1: ” si “Polinom 2: ”) polinomul/ polinoamele pe care vrea sa efectueze operatia.

**Atentie!**

In cazul in care utilizatorul nu va introduce al doilea polinom pentru operatiile pe doua polinoame va fi intampinat de un astfel de mesaj:



In cazul in care polinomul nu respecta formatul “(coeficient)x^(putere)+(coeficient)x^(putere)”, utilizatorul va fi atentionat:



In acest exemplu, mesajul se datoreaza faptului ca “x” a fost inlocuit cu “y”.

Pas4: Utilizatorul va apasa pe butonul corespunzator operatiei alese. In acest moment, rezultatul va fi afisat in campul de sub “Polinom 2: ”. In cazul in care utilizatorul va dori sa efectueze alta operatie pe cele 2 polinoame, trebuie doar sa apese butonul corespunzator operatiei dorite iar rezultatul se va schimba automat. In cazul debifarii “Operatii pe doua polinoame” atat continutul Polinomului 2 cat si cel al rezultatului va fi sters.

**7. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare**

Cu ajutorul acestei teme am reusit sa reiau conceptele programarii orientate pe obiect, sa invat noi metode de implementare ale unei interfete grafice si noi metode de parcurgere/ manipulare a unei liste.

Ca si dezvoltari ulterioare se pot face:

1.Rescrierea mai eficienta a operatiilor(metodele actuale de parcurgere/ manipulare nefiind cele mai eficiente).

2.Scrierea si afisarea polinoamelor intr-un format mult mai usor pentru utilizator – in cazul in care exista in polinom x^1 sa se afiseze doar x, coeficientii daca sunt intregi, sa nu mai fie afisati cu tipul float, in momentul in care exista un coeficient pe langa x^0, x^0 sa nu mai fie afisat/ necesar scrierii inainte de procesare.

3.Adaugarea de noii operatii cum ar fi gasirea radacinilor polinomului, sau efectuarea operatiilor de baza dar pe mai multe polinoame.

4,Interfata poate fi realizata mai user-friendly.

5.In momentul in care vrei sa executi o operatie pe rezultatul obtinut, acesta sa fie salvat automat in polinomul 1.

**8.Bibliografie**

<https://www.geeksforgeeks.org/iterators-in-java/>

<https://junit.org/junit4/javadoc/4.8/org/junit/Assert.html>

<https://www.emathhelp.net/calculators/algebra-1/polynomial-calculator/>