\

Facultatea de Automatică și Calculatoare

Tehnici de programare fundamentale

Laborator – Tema 1

Calculator Polinomial

Profesor: Dr. Cristina Pop

Student: Talpoș Andreea Georgiana

Grupa: 30228

1. Obiectivul temei

Obiectivul temei este proiectarea și implementarea unui calculator care să realizeze operații matematice între polinoame, având o interfață grafică corespunzătoare. Prin intermediul interfeței grafice, utilizatorul introduce cele două polinoame, selectează operația care urmează să fie executată și îi este returnat rezultatul. Operațiile posibile sunt: adunarea, scăderea, înmulțirea, împărțirea, derivarea și integrarea.

Se consideră doar polinoame de o singură variabilă și cu coeficienți întregi.

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

În matematică, un polinom este o expresie contruită dintr-un sau mai multe variabile și constante, fiind folosite doar operațiile de adunare, scădere, înmulțire și ridicare la putere constantă pozitivă întreagă. Polinoamele sunt construite din termeni numiți monoame, care sunt alcătuite dintr-o constantă (numită coeficient) inmulțită cu una sau mai multe variabile. Fiecare variabilă poate avea un exponent constant întreg pozitiv. Coeficientul unui monom poate fi orice număr. De exemplu 3x2 este un monom, coeficientul său este 3, x este variabila iar 2 este exponentul. Mai multe asemenea monoame adunate formează un polinom, de exemplu: 3x2 - 2x + 4 este un polinom având trei termeni cu exponenți diferiț: primul său termen este de grad 2, al doilea de grad 1, iar gradul celui de-al treilea este 0.

Orice polinom cu o singură variabilă x, poate fi scris sub forma:

a0 + a1x + a2x2 + .. + anxn =

Acest mod de reprezentare al polinoamelor poate fi folosit pentru a executa calcule comune precum adunarea, scăderea, înmulțirea, împărțirea, derivarea, integrarea etc.

1. Modelarea problemei

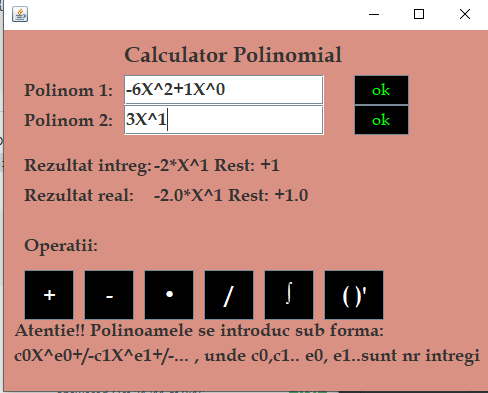
Utilizatorul va putea utiliza funțiile calculatorului introducând prin intermediul interfeței două polinoame. Acesta va trebui să valideze cele două polinoame pentru ca aplicația să poată stoca valorile, iar după va putea alege operația care dorește să fie executată:

* Adunarea a două polinoame
* Scăderea a două polinoame
* Înmulțirea a două polinoame
* Împărțirea a două polinoame
* Integrarea unui polinom
* Derivarea unui polinom

Rezultatul corespunzător, atât cel cu coeficienți întregi, cât și cel cu coeficienți reali, va fi afișat în interfață.

1. Scenarii si cazuri de utilizare

Scenariile și cazurile de utilizare sunt metode folosite in analiza sistemelor pentru a identifica, clarifica și a organiza cerințele sistemului. Un caz de utilizare reprezintă un set de posibile interacțiuni dintre utilizator și aplicație. Deoarece cazurile de utilizare reprezintă, de fapt, pașii pe care utilizatorul îi are de parcurs, am încercat să proiectez o interfață cât mai ușor de utilizat (“User Friendly”).



Utilizatorul introduce cele două polinoame în câmpurile corespunzătoare, apoi, apasă butoanele “ok” din dreptul fiecărui operand, pentru ca aplicația să le poată prelua și genera intern. Dacă dorește să efectueze o anumită operație, trebuie să apese butonul pe care este reprezentat grafic semnul operației. Va fi returnat atât rezultatul cu coeficienți întregi, cât și cel cu coeficienți reali.

Polinoamele trebuie introduse în formatul precizat în GUI, altfel, rezultatul nu va fi cel corect.

Diagrama use-case:

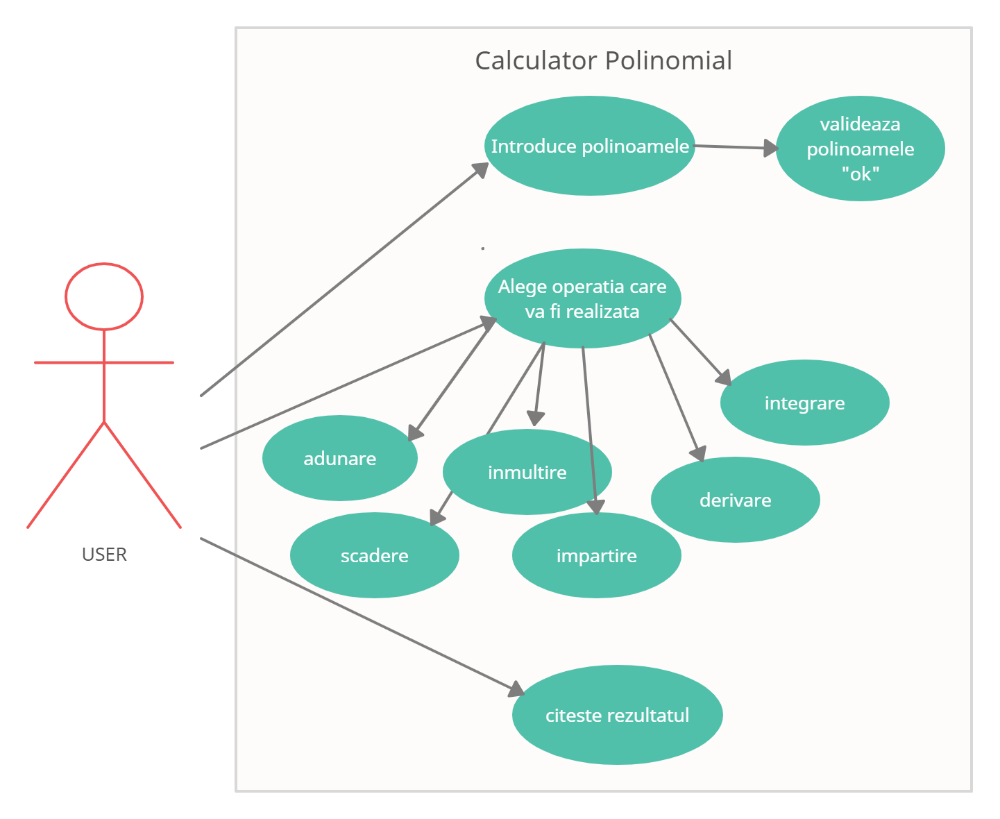


Diagrama use-case prezintă utilizatorul care interacționează cu aplicația. Acesta poate efectua mai multe operații cu polinoamele introduse: adunare, scădere, înmulțire, împărțire, derivare, integrare, și îi este afișat rezultatul.

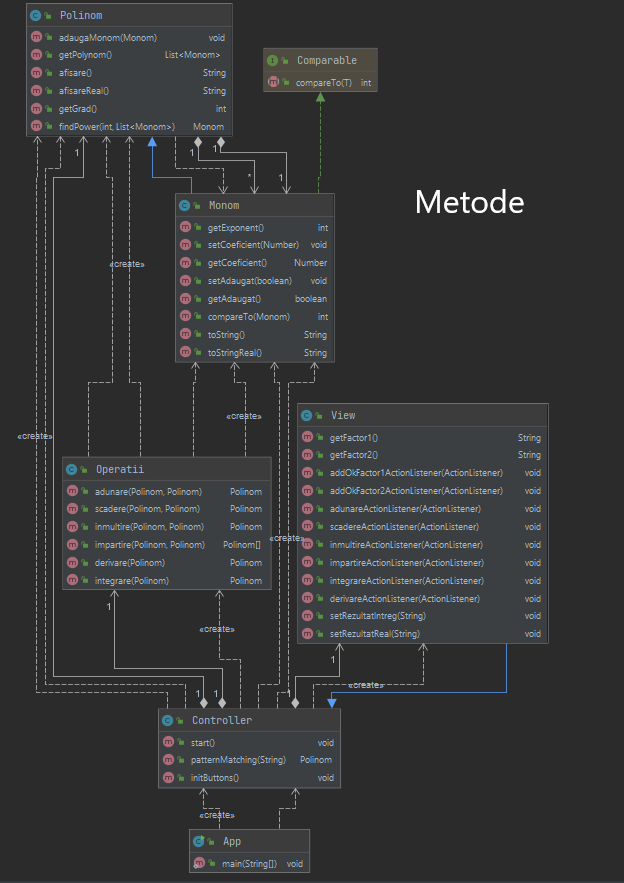
1. Proiectare

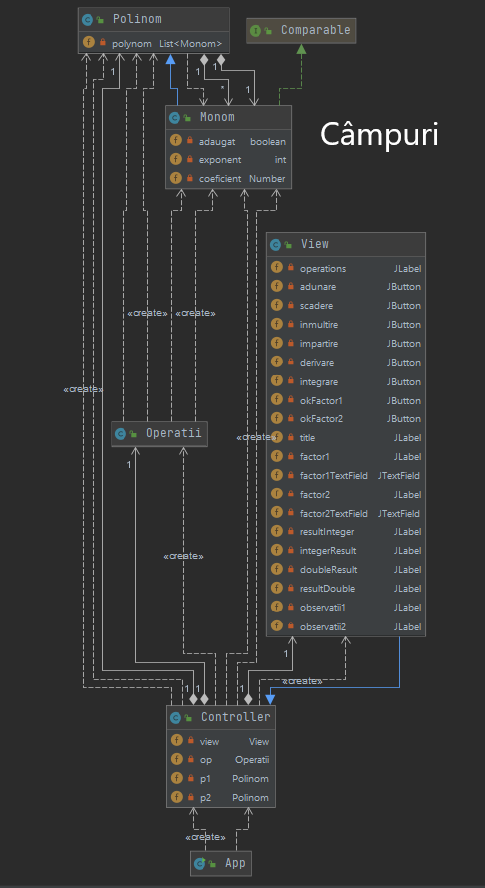
În ceea ce privește proiectarea aplicației, după analiza cerinței am concluzionat că proiectul meu trebuie să poată să preia două polinoame și operația pe care un presupus utilizator vrea să o efectueze asupra acestora, și apoi să returneze rezultatul operației respective. Am ales să organizez proiectul în două pachete: Model și GUI.

Pachetul Model conține clasele care modelează datele aplicației: Polinom și Monom și clasa Operații în care sunt implementate operațiile matematice dintre polinoame.

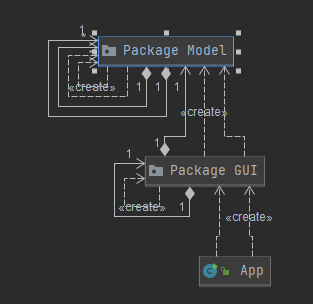
Pachetul GUI are două clase: View și Controller care implementează interfața grafică.

1. Diagrame UML de clase





1. Diagrama de pachete



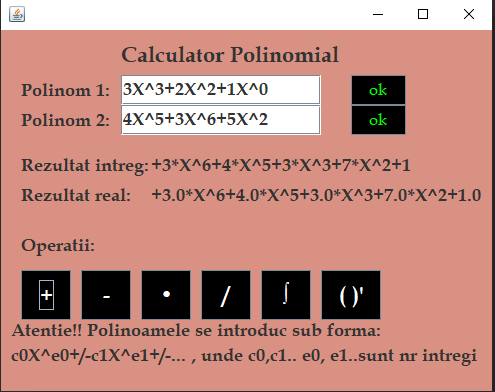
1. Structuri de date

Tipurile de date cu care am lucrat în acest proiect sunt, în mare parte, tipurile Integer și Number, și unele structuri mai complexe precum ArrayList și List de obiecte sau tipuri de obiecte noi create: Monom și Polinom. Am preferat să folosesc liste în locul vectorilor clasici deoarece sunt mai eficiente în ceea ce privește memoria ocupată, performanța, asigură un acces mai rapid la conținut și sunt mai ușor de actualizat, nu au o lungime fixă.

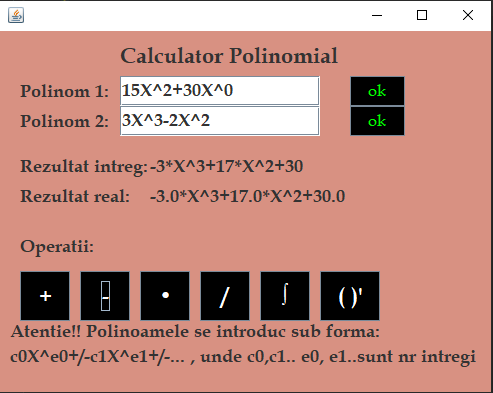
1. Interfața cu utilizatorul

**Exemple de operații efectuate:**

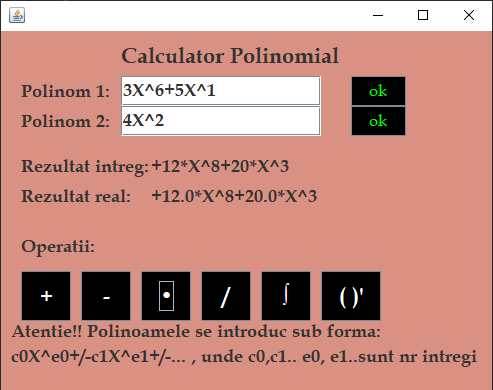
*Adunarea :*



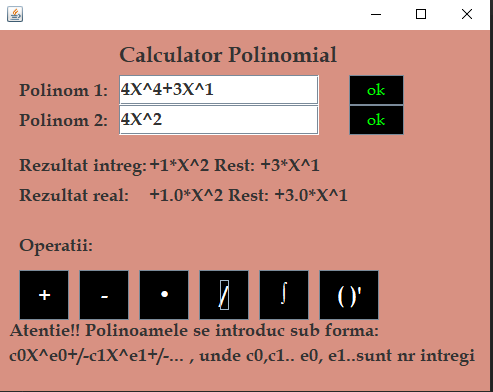
*Scăderea :*



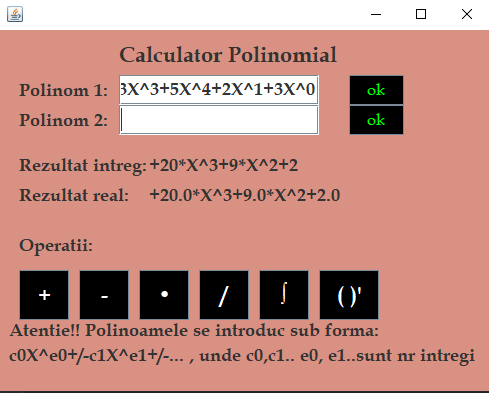
*Înmulțirea:*



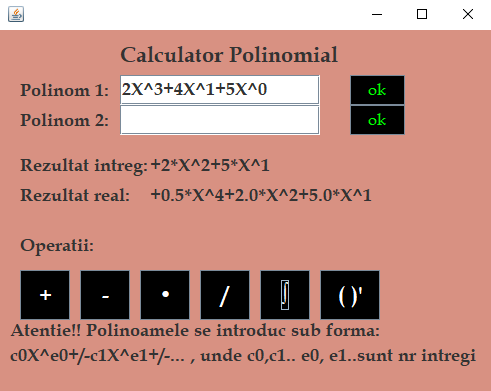
*Împărțirea :*



*Derivarea:*



*Integrarea :*



1. Implementare
2. Pachetul Model – conține logica aplicației
3. **Clasa Monom**

Un polinom este compus din unul sau mai mulți termeni numiți monomi. Clasa are trei variabile instanță, un boolean adăugat – am avut nevoie de el pentru a marca faptul că un monom a fost adăufat într-un polinom, un int exponent și coeficient de tip Number.

Constructorul:

*public Monom(int exp, Number coef) {*

*exponent = exp;*

*coeficient = coef;*

*}* -Inițializează monomul cu exponentul si coeficientul transmiși ca parametrii

Metode:

*-public int getExponent()* – returnează exponentul (gradul) monomului

- *public void setCoeficient(Number coeficient)* – ne permite să setăm coeficientul unui monom

- *public Number getCoeficient()* – returnează coeficientul monomului

- *public void setAdaugat(boolean adaugat)* – modifică valoarea variabilei instanță (true/false)

- *public boolean getAdaugat(*) – returnează valoarea variabilei “adaugat”

- *public int compareTo(Monom m)* – ordonez monomii descrescător în funție de exponent

*- public String toString()* – returnează monomul sub formă de string, cu coeficienții întregi

- *public String toStringReal()* - returnează monomul sub formă de string, cu coeficienții reali

1. **Clasa Polinom**

Clasa are doar o variabilă instanță, o listă de monoame, componente ale polinomului : private List<Monom> polynom = new ArrayList<Monom>().

Constructorul: - doar constructorul implicit

Metode:

-*public List<Monom> getPolynom()* - returnează polinomul propriu-zis, ca o listă de monoame

- *public String afisare()* – afișează polinomul cu coeficienți întregi, sub formă de string

- *public String afisareReal()*- afișează polinomul cu coeficienți reali, sub formă de string

- *public int getGrad()* – returnează gradul polinomului, adică exponentul maxim

- *public Monom findPower(int degree, List<Monom> poly)* – returnează monomul care are exponentul specificat, element al polinomului

1. **Clasa Operatii**

În această clasă am implementat operațiile care se fac între polinoame: adunare, scădere, înmulțire, împărțire, integrare și derivare. Nu are nicio variabilă instanță.

Metode:

- *public Polinom adunare(Polinom p, Polinom q)* – realizează adunarea dintre două polinoame

- am declarat un nou Polinom în care am adăugat pe rând monomii rezultași în urma adunării

- se adună coeficienții monomilor care au același exponent și se adaugă în polinomul rezultat, iar cei rămași ( care nu au corespondent în celălalt polinom, care să aibă aceeași putere) se adaugă în rezultat fără a suferi modificări

- *public Polinom scadere(Polinom p, Polinom q)* – realizează scăderea celor două polinoame

-e implementată la fel ca adunarea, doar că aici se scad coeficienții monomilor cu același exponent

- monomii din al doilea polinom ( scăzătorul) care nu au fost procesați, se adaugă cu semn schimbat în rezultat

- *public Polinom inmultire(Polinom p, Polinom q) -* realizează înmulțirea

- în polinomul rezultat am salvat monomii obținuți în urma înmulțirii fiecărui monom din primul polinom cu fiecare monom din al doilea, înmulțirea presupune adunarea exponenților și înmulțirea coeficienților

- în plus, am adunat monomii care aveau același exponent, din polinomul rezultat

- *public Polinom[] impartire(Polinom p, Polinom q)* -realizează împarțirea celor două polinoame

- în cazul împărțirii trebuie returnate două polinoame : câtul și restul

- două polinoame nu se pot împărți dacă cel de-al doilea are grad mai mare sau dacă este egal cu zero, am verificat dacă cumva suntem într-una din cele două situații, caz în care va apărea pe ecran un mesaj de atenționare

- am tratat separat situația în care al doilea polinom este de gradul 0, caz în care fiecare coeficient al monomilor din primul polinom se împarte cu constanta care formează al doilea polinom

- am initializat polinomul rezRemainder ( care va conține restul) cu valoarea primului polinom, cat timp restul are grad mai mare decât al doilea polinom, primul monom din rest se va împărți cu primul monom din al doilea polinom, monomul rezultat se înmulțește cu al doilea polinom (q) și rezultatul obținut se scade din rest

- *public Polinom derivare(Polinom p)* – realizează derivarea

- coeficientul fiecărui monom cu gradul mai mare decât 0 se înmulțește cu exponentul său, iar din exponent se scade 1

- *public Polinom integrare(Polinom p)* – integrarea polinomului

- se obține rezultatul corect doar pentru polinoamele cu exponenți pozitivi

- pentru fiecare monom se înmulțește coeficientul cu exponentul și se împarte cu exponentul adunat cu 1, iar la exponent se adună 1

1. Pachetul GUI
2. **Clasa View**

Această clasă implementează interfața grafică cu utilizatorul. Interfața grafică reprezintă un model pentru acea parte a unui program care interacționează cu utilizatorul și folosește pictograme pentru a reprezenta caracteristicile programului. Am folosit o interfață grafică ușor de utilizat cu scopul de a putea fi folosit acest calculator de polinoame si de persoanele nespecializate.

View extinde clasa Controller, în care am instanțiat un obiect la clasei View, și am implementat metodele necesare pentru a realiza legătura dintre interfață și program.

Pentru realizarea interfeței grafice am folosit pachetele java.awt și javax.swing care conțin clase și metode necesare pentru crearea GUI.

**Constructorul clasei:**

*- public View()*  - inițializează “rama” ( frame) și setează caracteristicile acesteia. De asemenea, plasează toate etichetele (labels), câmpurile de text și butoanele declarate ca variabile instanță

**Metode:**

- *public String getFactor1()* – metoda returnează polinomul introdus de utilizator în primul câmp de text ( polinomul 1)

- *public String getFactor2()* - metoda returnează polinomul introdus de utilizator în al doilea câmp pentru text ( polinomul 2 )

- *public void setRezultatIntreg(String rezultat)* – metoda setează textul label-ului pentru rezultatul întreg ca fiind string-ul obținut după efectuarea operației, reprezentând polinomul cu coeficienți întregi

- public void setRezultatReal(String rezultat - setează textul label-ului pentru rezultatul real, adică string-ul obținut după efectuarea operației, reprezentând polinomul cu coeficienți reali

Clasa conține de asemena și metode ActionListener pentru toate butoanele interfeței ( + ,- ,/ ,\* ,()’ ,∫ , ok – polinom 1, ok – polinom 2). Aceste metode sunt implementate în clasa Controller.

1. **Clasa Controller**

Clasa Controller face legătura dintre Model și interfața grafică. Este foarte importantă deoarece controlează transferul de date dintre logica internă și GUI, și actualizează interfața de fiecare dată când apare o modificare asupra datelor.

Are patru variabile instanță - private View view;

- private Operatii op = new Operatii();

- private Polinom p1;

- private Polinom p2;

Prin intermediul acestora accesează atât View, cât și polinoamele, pe care le controlează.

**Metode:**

*public void start()*{

view = new View();

view.setVisible(true);

initButtons();

}

* Prin intermediul acestei metode este activata interfața cu utilizatorul

*public Polinom patternMatching(String input){*  
 Pattern p =Pattern.*compile*("(-?\\d+)[xX]\\^(\\-?\\d+)");  
 Matcher m = p.matcher(input);  
 Polinom poly = new Polinom();  
 Monom monom;  
 int exponent;  
 Number coeficient;  
 while(m.find()){  
 int xPos = m.group(0).indexOf("X");  
 coeficient = Double.*parseDouble*(m.group(1).substring(0,xPos));  
 exponent = Integer.*parseInt*(m.group(2).substring(0));  
 monom = new Monom(exponent, coeficient);  
 poly.adaugaMonom(monom);  
 }  
 return poly;  
}

* Metoda patternMatching realizează conversia String-ului input primit ca parametru
* Folosesc această metodă pentru a prelua textul introdus de user pe care îl transform într-un polinom care să poată fi manipulat de către aplicație
* Există un pattern pe care un input introdus corect trebuie să îl respecte
* Știu că înainte de poziția lui “X” trebuie să se afle coeficientul unui monom, iar după semnul “^” se găsește exponentul,
* Am instanțiat un monom care va avea exponentul și coeficientul extrase din șirul de caractere primit ca parametru, și l-am adăugat într-un polinom, care va fi returnat la final, după adăugarea tuturor monoamelor identificate

*- public void initButtons()* – această metodă inițializează butoanele, implementează metodele ActionListener din View.

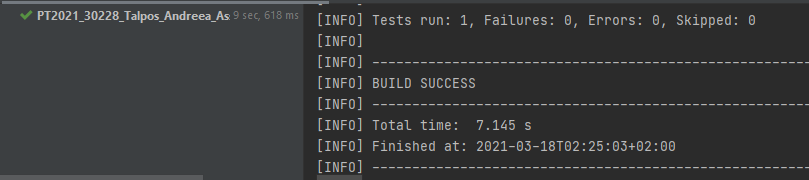
**Clasa App** din proiect instanțiază un obiect de tipul Controller cu care apelează metoda start() care va porni aplicația.

public class App   
{  
 public static void main( String[] args )  
 {  
 Controller control = new Controller();  
 control.start();  
 }  
}

1. Rezultate :

Am realizat testarea operațiilor dintre polinoame cu Junit.

|  |  |  |  |
| --- | --- | --- | --- |
| Operația | Intrarea | Rezultatul așteptat | Rezultatul obținut |
| Adunare | P : +1\*X^4+2\*X^2+5  Q : +1\*X^2+1 | +1\*X^4+3\*X^2+6 | +1\*X^4+3\*X^2+6 |
| Scădere | P : +1\*X^4+2\*X^2+5  Q : +1\*X^2+1 | +1\*X^4+1\*X^2+4 | +1\*X^4+1\*X^2+4 |
| Înmulțire | P : +1\*X^4+2\*X^2+5  Q : +1\*X^2+1 | +1\*X^6+3\*X^4+7\*X^2+5 | +1\*X^6+3\*X^4+7\*X^2+5 |
| Împărțire | P : +1\*X^4+2\*X^2+5  Q : +1\*X^2+1 | Cât: +1\*X^2+1  Rest: +4 | Cât: +1\*X^2+1  Rest: +4 |
| Integrare | P : +1\*X^4+2\*X^2+5 | +4\*X^3+4\*X^1 | +4\*X^3+4\*X^1 |
| Derivare | P : +1\*X^4+2\*X^2+5 | +0.2\*X^5+0.6666667\*X^3+5.0\*X^1 | +0.2\*X^5+0.6666667\*X^3+5.0\*X^1 |



1. Concluzii

Acest proiect m-a ajutat să aprofundez conceptele POO învățate în decursul semestrului trecut. De asemenea, am învățat multe lucruri noi legate de implementarea și proiectarea GUI prin intermediul documentării pe care a trebuit să o fac pentru a putea înțelege cum se realizează diferite oeprațiuni. Semestrul trecut am făcut doar programe foarte simple care aveau GUI. Mi-a fost mai greu când am început să lucrez la acest proiect, deoarece aveam de clarificat unele concepte, dar am găsit informații foarte utile, am urmărit chiar și video-uri și până la urmă, zic eu, a ieșit ceva destul de bun.

În ceea ce privește o dezvoltare ulterioară, cred că există și implementări mai eficiente decât cele propuse de mine, și ar aduce un plus aplicației.

1. Bibliografie
2. <https://creately.com>
3. <https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/Assertions.html>
4. <https://www3.ntu.edu.sg/home/ehchua/programming/java/j4a_gui.html>
5. <https://regex101.com>
6. [www.stackoverflow.com](http://www.stackoverflow.com)
7. <https://www.wikipedia.org>
8. <https://www.youtube.com>