

NUME : COSTEA PAULA MARIA

GRUPA : 302210

PROFESOR LABORATOR : DAN MITREA

**ASSIGNMENT 1**

**POLYNOMIAL CALCULATOR**

CUPRINS

1. Obiectivul temei

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

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

4. Implementare

5. Rezultate

6. Concluzii

7. Bibliografie

1. Obiectivul temei

Obiectivul principal al acestei temei este proiectarea si implementarea unui calculator polinomial cu o interfata grafica dedicata prin care utilizatorul poate insera polinoame, selecta operatia matematica dorita (adunare, scadere, inmultire, impartire, derivare, integrare) si vizualiza rezultatul.

Printre obiectivele secundare ale acestei teme se numara:

* Descrierea clasei „ Monom ” : reprezinta clasa de baza cu ajutorul careia realizam restul claselor;
* Descrierea clasei „ Polinom ” : reprezinta un sir de monoame, care de fapt este chiar polinomul in sine;
* Descrierea clasei „ Operatii ” : reprezinta clasa unde sunt implementate toate operatiile pe care le poate efectua acest calculator dar si o metoda prin care un string este `transformat` intr-un polinom.
* Descrierea clasei „ CalcView ” : reprezinta interfata grafica, acea parte din program care interactioneaza cu utilizatorul si foloseste componente specifice pentru a reprezenta caracteristicile programului.
* Descrierea clasei „ CalcController ” : reprezinta clasa in care se i atribuie fiecarei componente mentionate in interfata grafica o functionalitate.

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

* Analiza

Scopul principal al acestei teme este de a implementa o aplicatie care poate executa sase operatii matematice cu polinoame utilizand paradigmele programarii orientate pe obiect. Pornim de la problema retinerii unui polinom, care stim ca reprezinta de fapt o lista de monoame.

Cel mai important aspect al acestui proiect este implementarea operatiilor, in care trebuie sa tinem cont si de cerintele minime ale unei programari eficiente.

* Modelare

Unul dintre cele mai importante aspecte de retinut este faptul ca utilizatorul va introduce intotdeauna informatia pe care acesta doreste sa o proceseze sub forma de text ( string ). Noi va trebui sa transformam acest sir de caractere in ceva mai concret, cu care vom putea lucra ulterior.

Aceasta adaptare a informatiei o vom face pe baza faptului ca, cunoastem structura specifica a unui polinom cu care vom lucra, si mai exact, stim ca: polinomul reprezinta o lista de monoame iar un monom este reprezentat de doua atribute: putere si coeficient.

* Scenariul

Pentru ca programul sa functioneze corect si sa returneze rezultatul asteptat in urma unei

actiuni corespunzatoare, este necesar ca utilizatorul sa urmeze un anumit set de reguli, precum : necunoscuta pentru polinomul introdus trebuie sa fie reprezentata de litera „ X ” iar un termen din acesta trebuie sa fie de forma „ coefX^putere ”. Termenii polinomului nostru sunt delimitati de semnul “ + ”. In cazul in care se doreste adaugarea unui monom cu coeficient negativ, atunci introducerea sa se va face sub forma +-coefX^putere ( ca de exemplu, pentru ca monomul 5X^2-3X^4 sa fie interpretat corespunzator, tastarea lui trebuie sa se realizeze sub forma : 5X^2+-3X^4), datorita mentiunii facute mai sus. De asemenea, pentru ca operatiile de derivare si integrare sa functioneze, trebuie retinut faptul ca polinomul asupra caruia se aplica operatia este cel din prima caseta text.

* Cazuri de utilizare

Cand utilizatorul ruleaza aceasta aplicatie, o interfata grafica se va deschide. Am

gandit design-ul in asa fel incat sa fie usor de inteles si facil de utilizat. Interfata contine doua TextField-uri in care se vor introduce de la tastatura polinoamele asupra carora se vor aplica operatiile mentionate mai sus.

Pe langa acestea, interfata mai contine si sase butoane denumite reprezentativ pentru fiecare operatie. Dupa ce a introdus polinoamele, utilizatorul va apasa pe unul dintre butoanele care reprezinta operatia pe care acesta doreste sa o execute. Rezultatul va fi afisat intr-un label, situat chiar mai jos de cele doua casete text. In plus, pe langa cele 6 butoane pentru operatii, se mai gaseste si un buton “ Clear ”, care reseteaza continutul casutelor text, adaugand sirul vid, fiind pregatite pentru a stoca rezultatul urmatoarei operatii pe care utilizatorul va dori sa o execute.

3. Proiectare

Acest program respecta o arhitectura clasica MVC ( Model View Controller ), fiind impartit in mai mute pachete. In pachetul „ Model ” se afla clasele : „ Monom ”, „ Polinom ” si „ Operatii ”, in pachetul „ Controller ” se afla clasa „ CalcController ”, in pachetul „ View ” se afla clasa „ View ” iar in pachetul „ MVC ” se afla clasa „ MainClass ”.

Clasa de baza a acestui program este clasa „ Monom ”, care contine:

-2 atribute ( putere, de tip int si coef, de tip double );

-Getters si Setters pentru cele doua atribute;

-un constructor.

Pornind de la aceasta clasa, am implementat clasa „ Polinom ”, care reprezinta o lista de monoame si contine un constructor gol. Pe langa acestea, aici se mai gasesc si urmatoarele metode:

-construirePol ( monom );

-construirePol1 (monom );

-construirePol2 ( monom );

-verif ( monom );

-gradMax ( );

-maxim ( );

-toString ( );

Clasa „ Operatii ” contine implementarea algoritmilor pentru efectuarea operatiilor asupra polinoamelor, fiind descrisi prin metodele: adunarePoli ( p1, p2 ), scaderePoli ( p1, p2 ), inmultirePoli ( p1, p2 ), impartirePoli ( p1, p2 ), derivarePol ( p1 ), integrarePol ( p1 ). Pe langa acestea, se mai gasesc inca doua metode: metoda auxiliar(p1), care copiaza continutul polinomului p1 si metoda formarePolinom ( sir ), care transforma sirul de caractere introdus de utilizator intr-un polinom.

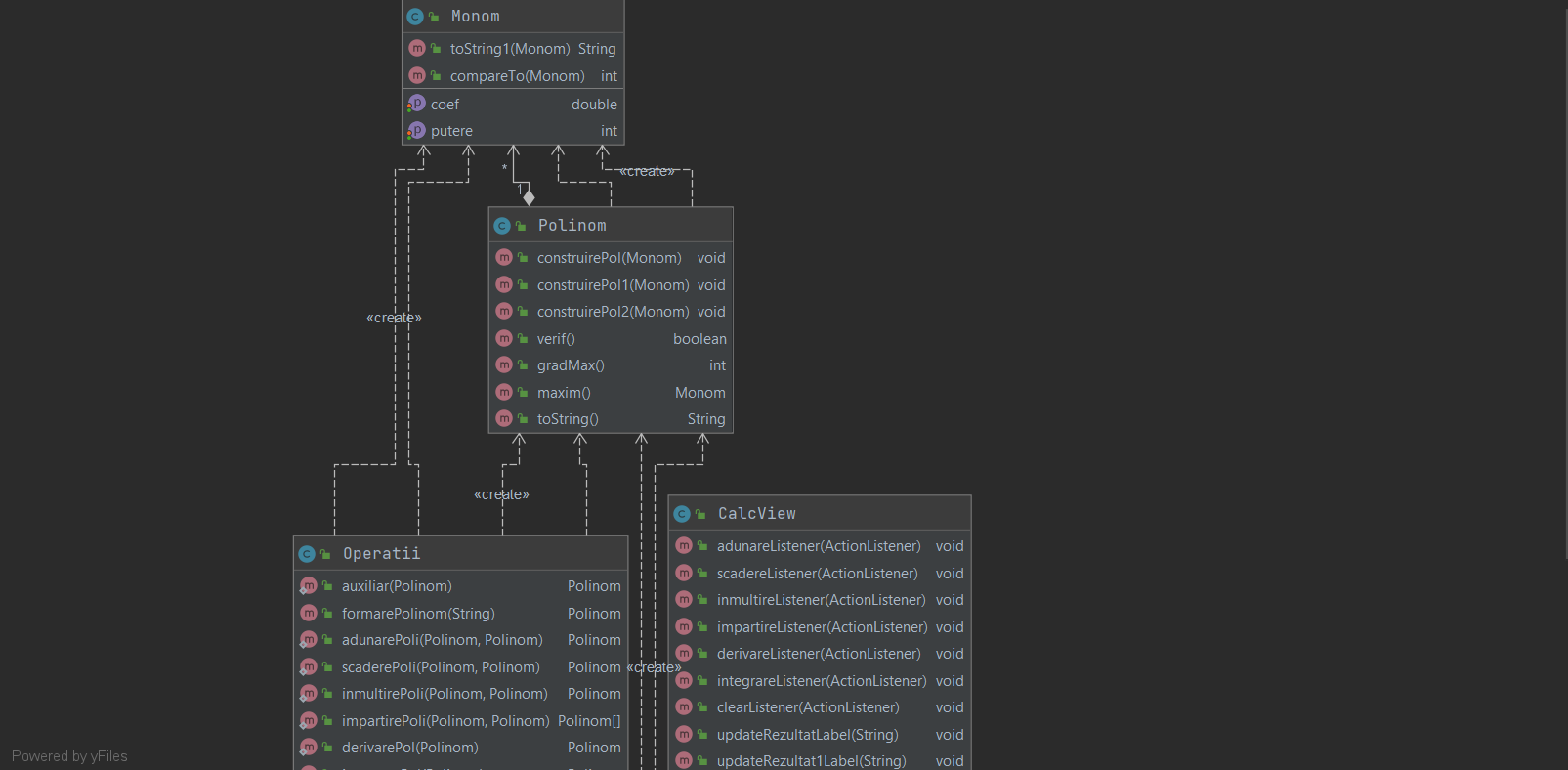
Clasa „ CalcView ” contine elementele de structura necesare pentru interfata grafica a proiectului. Aceasta face interactiunea cu computerul mai usoara si mai eficienta pentru utilizator. In implementarea ei, am utilizat componente precum: Jbutton, JLabel (componente atomice), JTextField (componente pentru editarea textului), JPanel (componente intermediare). Tot in acasta clasa declaram si modul in care aranjam toate aceste structuri, atribuindu-le fiecareia anumite anumite coordonate.

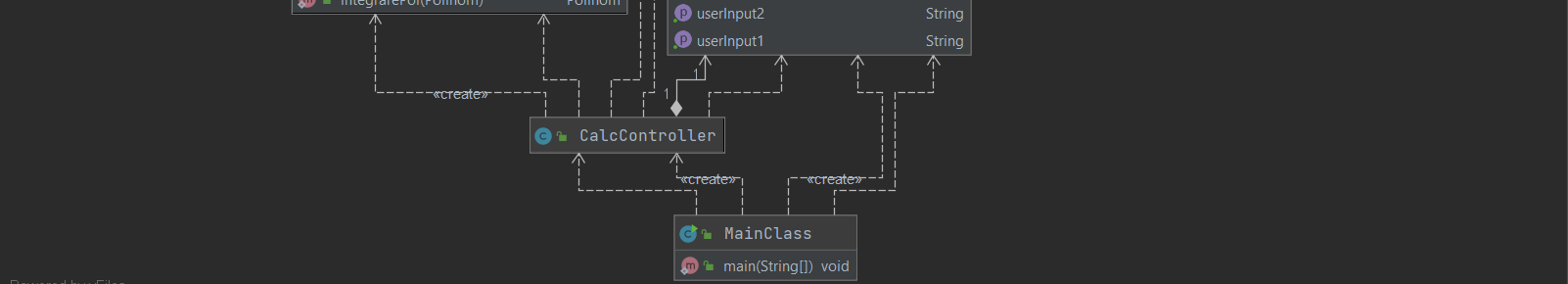
Clasa „ CalcController ” reprezinta clasa care traduce interacțiunile utilizatorului cu vederea în acțiuni pe care le va executa modelul.

Pentru operatiile descrise mai sus, am implemenat si o clasa de test, „ OperatiiTest ”, pentru a testa functionalitatea acestora, continand teste JUnit.

In final, clasa „ MainClass ” este clasa care leaga toate componentele descrise mai sus intre ele si ne ajuta sa rulam aplicatia.

Diagrama UML





4. Implementare

* Clasa „ Monom ” : aceasta clasa are doua atribute: putere ( int ) si coef ( double ). Tot aici se gasesc si getters si setters pentru aceste. Getters este metoda care returneaza valoarea coeficientului sau puterii monomului actual iar Setters seteaza o valoare puterii sau coeficientului pentru un anumit monom.
* Clasa „ Polinom ” : aceasta clasa are ca si atribut sirul de monoame, „pol”. Aici se mai gaseste si un constructor care ajuta la creearea polinoamelor. Pe langa acestea, aici sunt implementate diferite metode:
  + construirePol( Monom m1 ) : cu ajutorul acestei metode adaugam monomul m1 la polinomul dorit. Parcurgem polinomul in scopul gasirii unui monom cu acelasi grad cu al monomului dat ca si parametru. Am utilizat un „for” care imi incrementeaza un index, in care se va retine pozitia (indexul) monomului cautat dar si o variabila booleana pentru a verifica gasirea lui. In caz afimativ, adaugarea se face prin adunarea coeficientilor celor doua monoame, altfel adaugam monomul m1 la sfarsitul polinomului.
  + construirePol1( Monom m1 ) : aceasta metoda este analoaga cu cea prezentata mai sus dar in cazul afirmativ ( in cazul in care gasim un monom cu acelasi grad cu al monomului cautat) adaugarea se face prin scaderea coeficientilor.
  + construirePol2( Monom m1 ) : aceasta metoda inmulteste un polinom cu monomul m1 dat ca si parametru.
  + gradMax() : aceasta metoda cauta monomul cu gradul cel mai mare si returneaza gradul maxim.
  + maxim() : returneaza monomul cu gradul maxim dintr-un polinom.
  + verif(): aceasta metoda returneaza „true” in cazul in care polinomul dat este gol.
  + toString(): aceasta metoda trasforma un polinom intr-un sir de caractere. Am tratat trei cazuri ( coef<0, coef>0, coef=0 ) iar pentru fiecare dintre acestea se concateneaza sirul diferit.
* Clasa „ Operatii ” : este clasa in care realizam de fapt scopul acestui program prin implementarea operatiilor amintite mai sus. Pentru fiecare operatie este implementata o metoda separata:
  + formarePolinom ( String sir ) : am utilizat functia `split` pentru a-mi desparti string-ul cu ajutorul caracterului „ + ”. Acum, in sirul sir2 avem monoamele cu care vom forma un polinom. Am parcurs sirul sir2, iar pentru fiecare componenta am aflat coeficientul, folosindu-ne de indexul caracterului „X”, care reprezinta necunoscuta, si puterea, analog dar folosindu-ne de indexul caracterului „ ^ ”.
  + adunarePoli ( Polinom p1, Polinom p2) : mai intai copiem polinomul p1 in polinomul `rezultat` ( cel pe care urmeaza sa il returnam ) iar apoi utilizam metoda cosntruirePol( Monom monom) pentru a adauga pe rand fiecare monom al polinomului p2 la polinomul `rezultat`.
  + scaderePoli ( Polinom p1, Polinom p2) : mai intai copiem polinomul p1 in polinomul `rezultat` ( cel pe care urmeaza sa il returnam ) iar apoi utilizam metoda cosntruirePol1( Monom monom) pentru a adauga pe rand fiecare monom al polinomului p2 la polinomul `rezultat`.
  + inmultirePoli ( Polinom p1, Polinom p2) : mai intai inmultim fiecare monom al polinomului p1 cu fiecare monom al polinomului p2 iar apoi tratam cazul in care avem mai multe monoame cu aceeasi putere; parcurgem fiecare calup de astfel de monoame si ne folosim de variabila coeff ( in care adunam coeficientii ) pentru a determina coeficientul final al monomului cu o anumita putere.
  + impartirePoli ( Polinom p1, Polinom p2) : pentru a implementa aceasta operatia am folosit algoritmul „Diviziunea lunga polinomiala”, care este un algoritm pentru împărțirea unui polinom cu un alt polinom de același grad sau inferior. Diviziunea lungă polinomială este un algoritm care implementează diviziunea euclidiană a polinoamelor , care pornind de la două polinoame p1 și p2 produce, dacă B nu este zero, un coeficient q și un rest r astfel p1 = p2q+ r.
  + derivarePol (Polinom p1) : pentru a implementa aceasta metoda am utilizat formula

matematica specifica asupra fiecarui monom : ( coefX^putere ) ` = coef\*putereX^(putere-1).

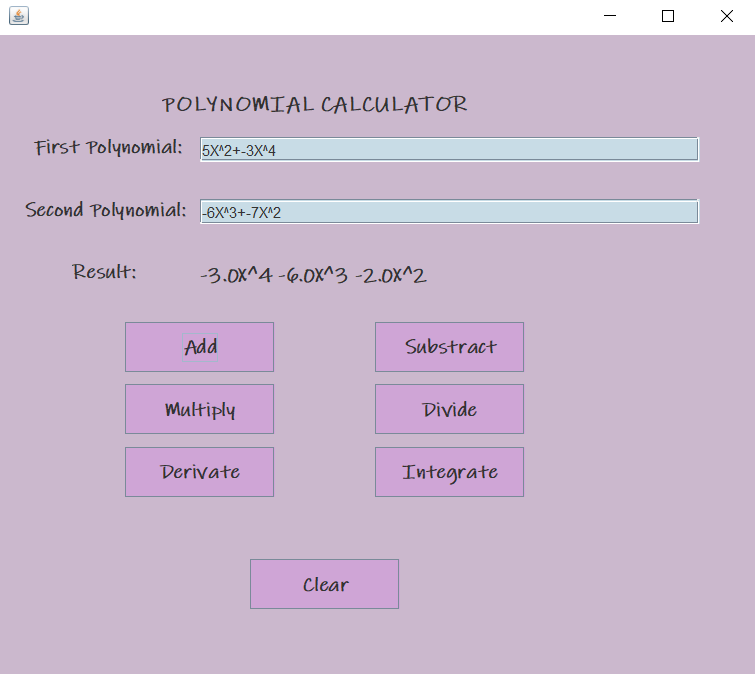
* + integrarePol (Polinom p1) : pentru a implementa aceasta metoda am utilizat formula

matematica specifica asupra fiecarui monom : = coef\*( 1 / ( n + 1 ))X ^ ( putere + 1 ).

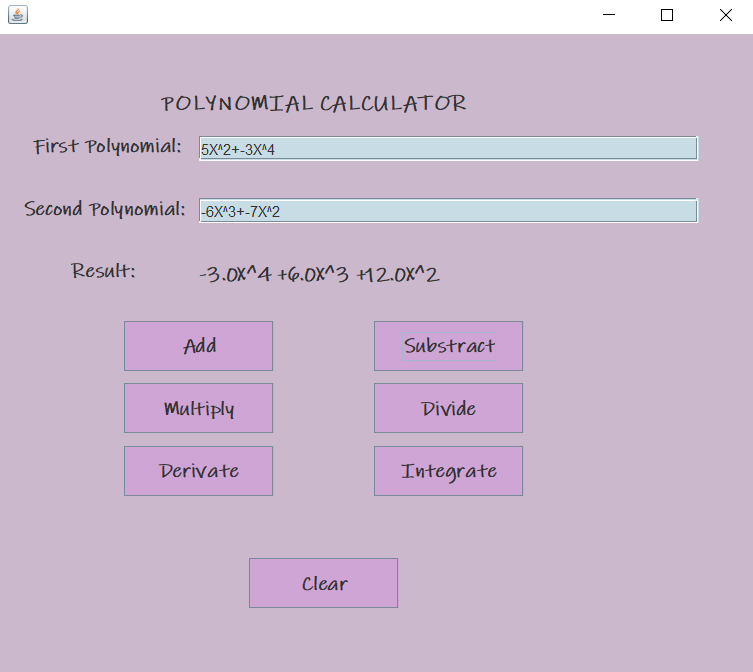
* Clasa „ CalcView ” : in aceasta clasa se afla toate componentele necesare in realizarea interfetei specifice cat si diferite metode, ca de exemplu, metode care ne ajuta sa culegem informatia din cele doua casete text sau prin care putem actualiza informatia din anumite label-uri. Tot in aceasta clasa, se gaseste pentru fiecare buton un ActionListener().
* Clasa „ CalcController” : aceasta este clasa care se ocupa de realizarea „evenimentelor” in momentul in care se apasa pe un anumit buton. Aici, pentru fiecare ascultator adaugat se realizeaza o anumita operatie a carei rezultat este afisat pe ecran.
* Clasa „ OperatiiTest” : in cadrul acestei clase am realizat testari individuale pentru fiecare operatie in parte pentru a verifica corectitudinea acestora.
* Clasa „ MainClass” : este clasa in care instantiem un obiect de tip CalcController dar si interfata grafica.

5. Rezultate

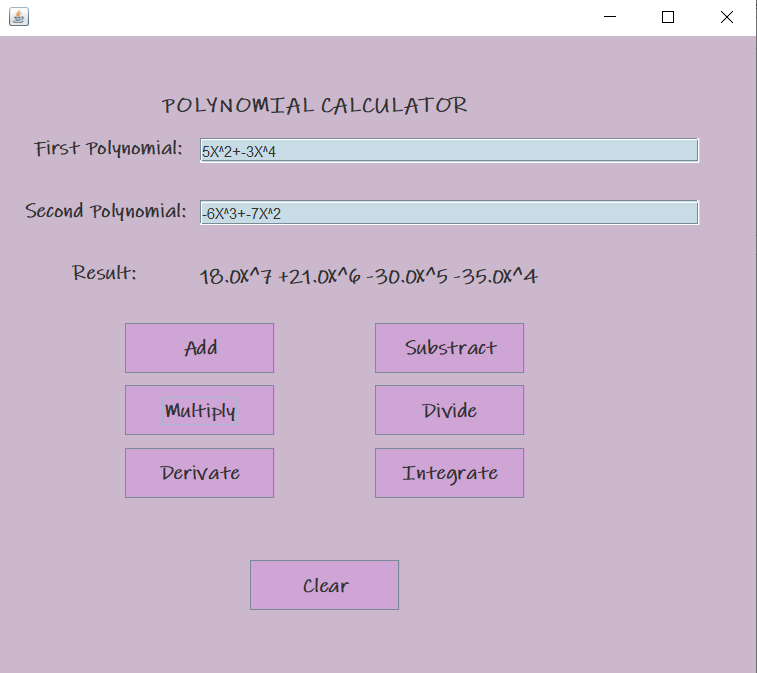
- Adunare :



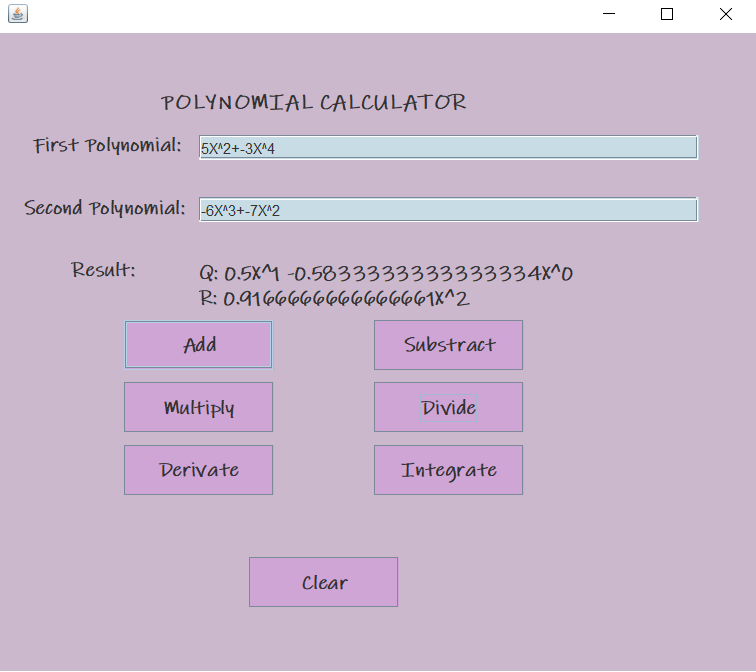
- Scadere :



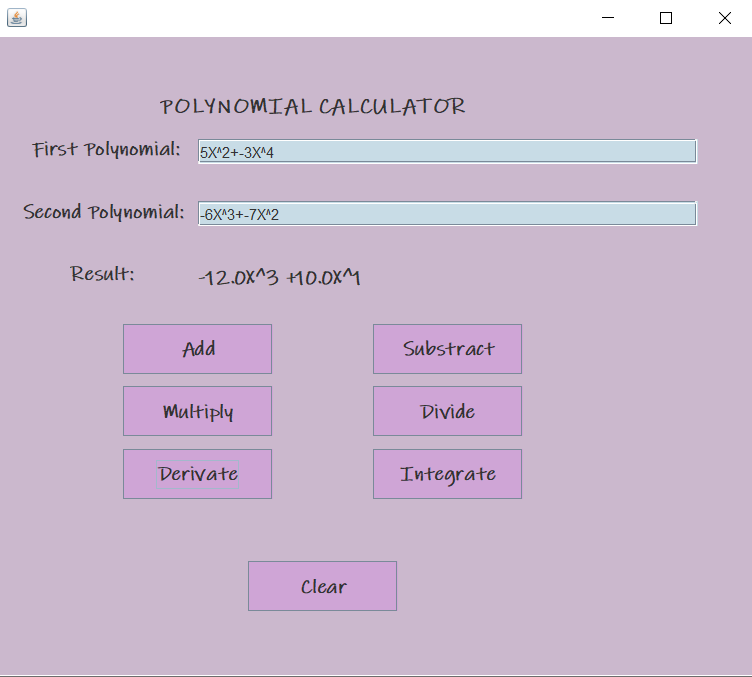
- Inmultire :



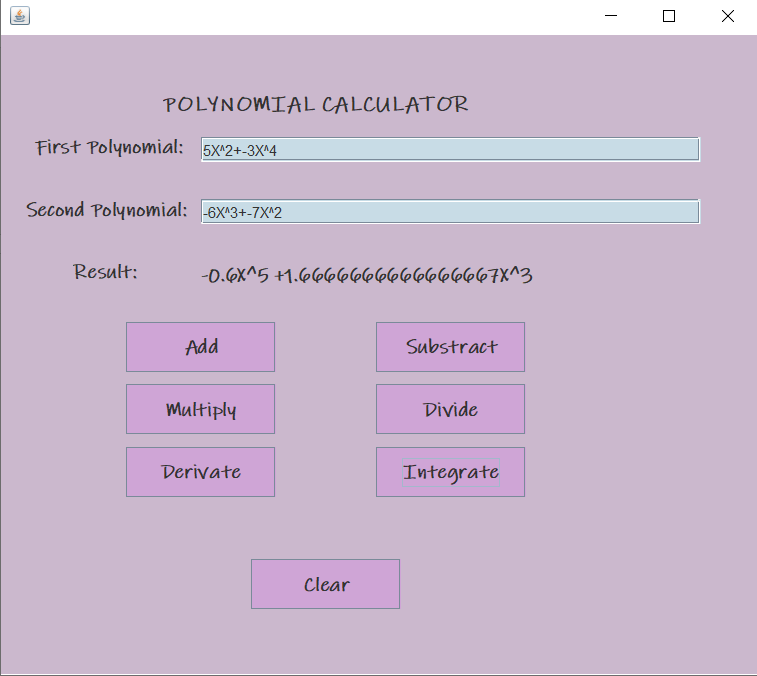
- Impartire :



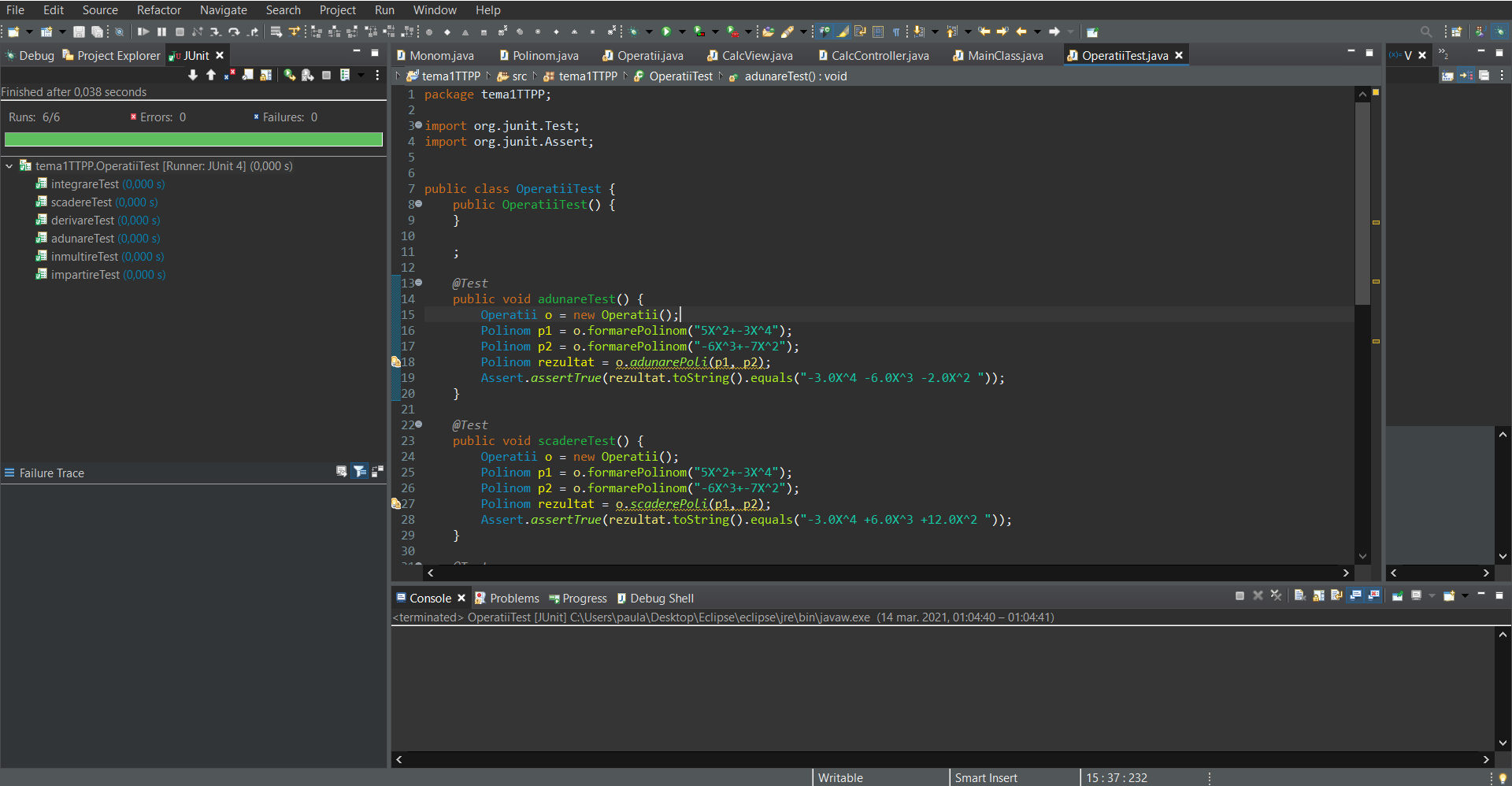
- Derivare :



- Integrare :



Testare



6. Concluzii

Dupa mine, aceasta prima tema a relizat o adevarata provocare deoarece am fost pusi in situatia de a implementa aceasta aplicatie de la zero. Una dintre greutatile intampinate pe parcurs a fost cea a convertirii unui string intr-un polinom, care de fapt reprezenta primul pas in a putea continua cu implementarea operatiilor, adevaratul scop al programului. M-am straduit sa implementez aceasta aplicatie respectand paradigmele programarii orientate pe obiect. Un avantaj al acestui calculator de polinoame este faptul ca utilizatorul ar putea aplica mai multe operatii consecutive asupra polinoamelor introduse intitial, fara a fi necesar sa le rescrie. As putea adauga faptul ca, o imbunatatire ulterioara ar putea reprezenta implementarea unei baze standard precum transformata Laplace.

7. Bibliografie

* <https://ro.qaz.wiki/wiki/Polynomial_long_division>
* <https://stackoverflow.com/>
* <https://ro.wikipedia.org/>
* <https://www.geeksforgeeks.org/>