

Documentație laborator tema 1

Calculator de polinoame

Tehnici fundamentale de programare

Student: Dunca Denisa Mihaela

Grupa: 30227

1. **Obiectivul temei**

Cerința temei este de a crea și implementa un calculator polinomial cu o interfață grafică dedicată prin care utilizatorul să poată să introducă polinoame, să selecteze o operație matematică (adunare, scădere, înmulțire, împărțire, derivare sau integrare) pentru a obține un rezultat corect la final.

Tema de laborator “Calculator de polinoame” are ca și obiectiv principal obținerea unui calculator care să dea rezultatul corect atunci când sunt aplicate anumite operații asupra a două polinoame pe care utilizatorul le introduce de la tastatură.

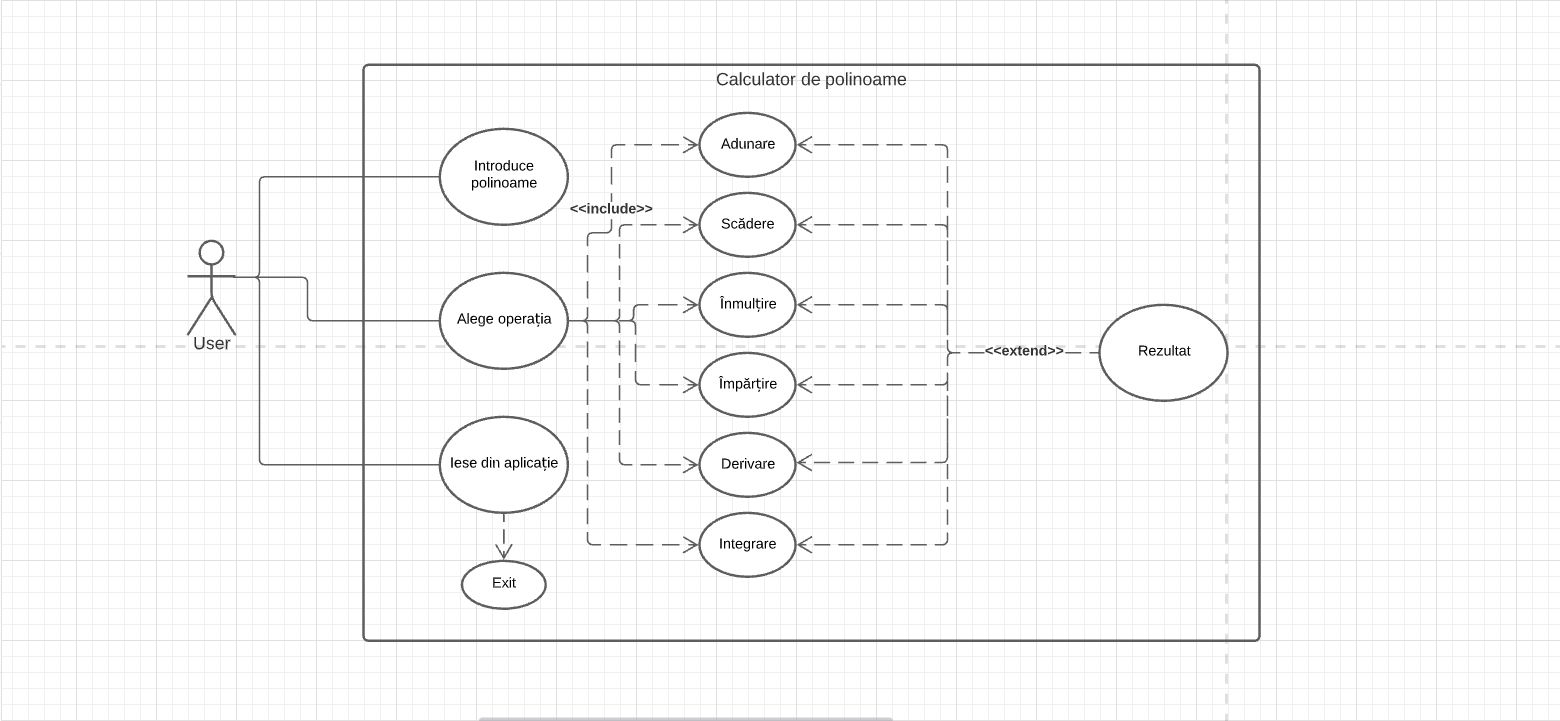
Obiectivele secundare ale proiectului:

* Analizarea problemei prin înțelegerea corectă a cerinței și modelarea de scenarii și cazuri de utilizare.
* Proiectarea unei soluții care să se plieze pe toate cazurile pe care le introduce utilizatorul.
* Implementarea soluției alese prin scrierea codului implicit și a interfeței utilizator.
* Observarea rezultatelor finale și concluziile.

1. **Analiza problemei**

Analiza problemei a constat prin înțelegerea pe deplin a cerinței și a informațiilor pe care dorim să le primim prin generarea unui rezultat final. S-a creat o schemă inițială care trebuie să prezinte în mare ceea ce se urmărește prin acest proiect. Pe de o parte, este necesară cunoașterea structurii unui polinom, astfel încât utilizatorul să introducă datele corecte pentru a i se genera răspunsul dorit. Pe de altă parte, programul trebuie să funcționeze pe structura: introducere cerință, calcul, generare răspuns, ieșire.

Un mod prin care se poate scoate în evidență această structură este generarea unei Diagrame USE CASE.



În diagrama de mai sus este prezentată structura logică a proiectului. Sistemul este un calculator de polinoame, user-ul reprezintă actorul, cel care efectuează operații, cel care dorește un răspuns. Introducerea polinoamelor este dependentă de actor, precum și alegerea operației dorite și ieșirea din program. Alegerea operație include mai multe variante dintre care autorul poate să aleagă, fiecare în parte generând răspunsul așteptat. Rezultatul este într-o relație de ”extend” cu operațiile deoarece depinde de acestea și se generează abia după ce una din ele a fost selectată.

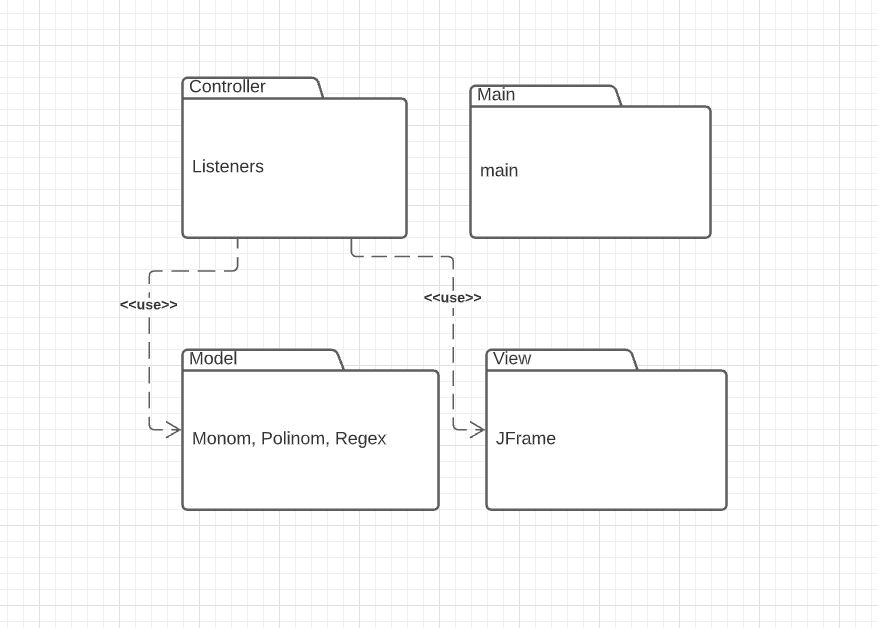
Un scenariu de utilizare ar fi:

* User-ul introduce primul polinom de la tastatură, ex: 2x^3 + 3x^2 - 2x^0
* Pe urmă, introduce cel de-al doilea polinom, ex: 4x^3 - 2x^1
* Selectează operația de adunare
* Pe ecran apare rezultatul adunării celor două polinoame introduse: 6x^3 + 3x^2 - 2x^1 - 2
* User-ul iese din aplicație

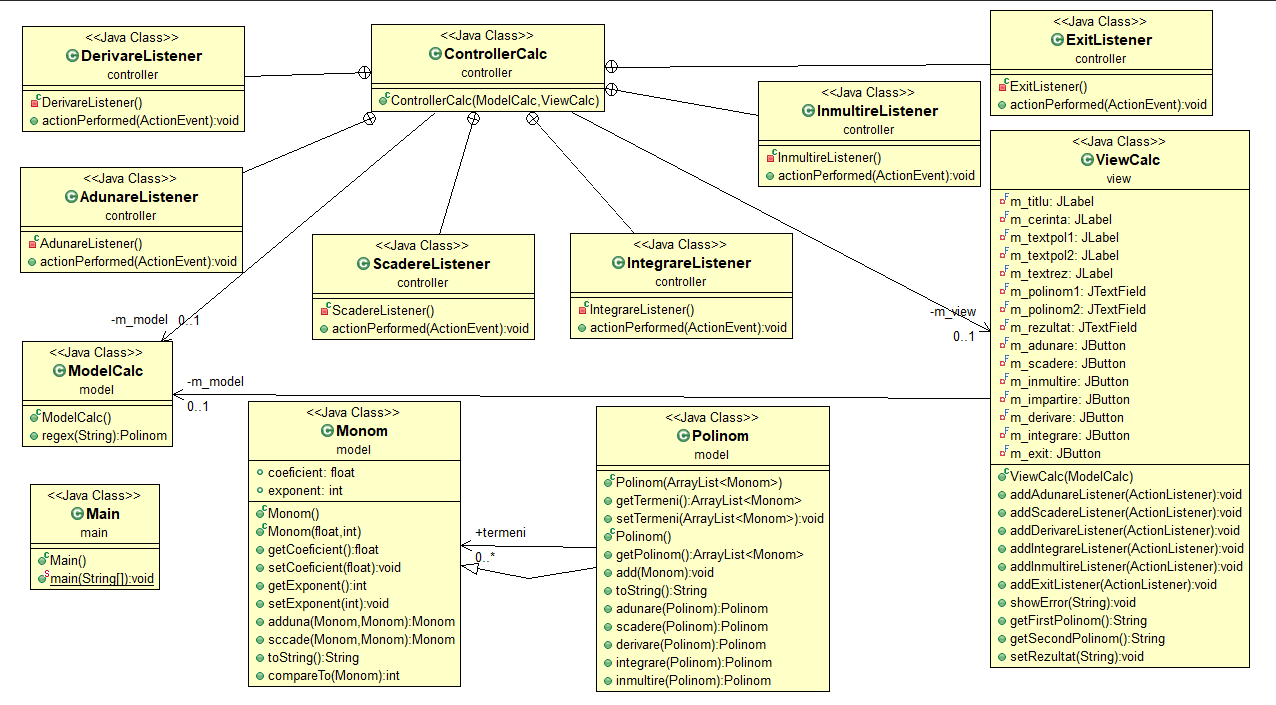
1. **Proiectare**

Trecând la partea practică de rezolvare a problemei, și anume proiectarea în limbaj OOP, au fost urmăriți mai mulți pași, primul fiind crearea unei Diagrame de pachete urmărind modelul MVC (Model-View-Controller).

Controllerul face legătura dintre clasele din Model și View prin intermediul butoanelor apăsate. În model se află operațiile și tot ce înseamnă funcționalitate a algoritmilor, iar în View partea ce va fi disponibilă utilizatorului spre folosire.



Intrând într-o analiză mai amănunțită a proiectului, am generat și diagrama UML. Aici se pot vedea în detaliu legăturile dintre clase, metodele folosite, precum și structura logică a întregului proiect.



Este respectată în principal structura MVC a proiectului. În View am declarat toate obiectele de care am nevoie folosind clasa JFrame, în Controller am creat un obiect de tipul Model, respectiv View și am creat listener-e pentru butoane. Modelul este alcătuit din trei clase, clasa Monom unde am definit un constructor ce are un coeficient și un exponent, tot aici am și clasa de toString pentru a converti polinomul rezultat în toString. Clasa Polinom are ca structură de date folosită, ArrayList pentru a stoca o lista monoame ce formează polinoame, tot aici am pus și toate metodele ce lucrează cu polinoame, operațiile, iar în clasa Model am metoda regex care verifică dacă este respectat pattern-ul dat.

1. **Implementare**

**Pachetul model**

**Clasa monom**

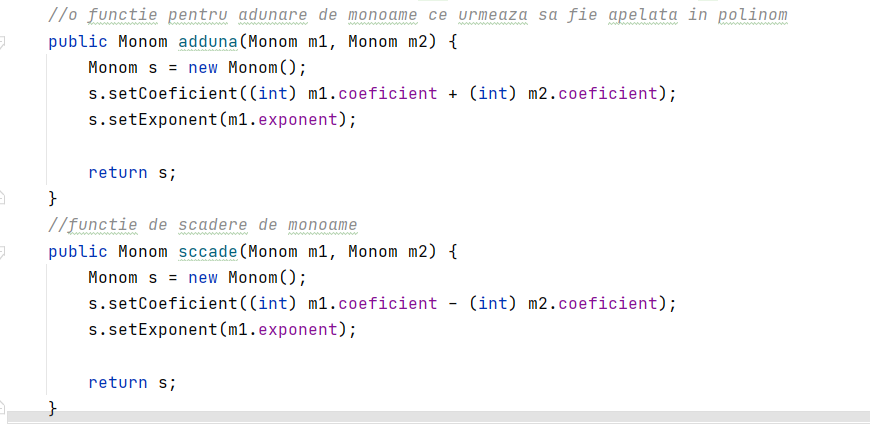
Clasa monom are ca obiectiv principal crearea unei structuri de forma: ax^b , unde ”a” este coeficientul, ”x” necunoscuta și ”b” exponentul. Clasa primește ca și variabile un float pentru coeficient și un int pentru exponent, apoi un constructor care să le conțină pe amândouă variabile. Pe lângă getter-e și setter-e am implementat și metoda compareTo specifică interfeței Comparable.

Metoda care ocupa cel mai mult din clasa este metoda toString cu ajutorul căreia putem afișa rezultatul nostru sub forma de string. Metoda presupune fiecare caz ce poate apărea, de aceea constă în multe if-uri imbricate și multe ramuri de else. Funcționalitate: se verifică prima data în funcție de exponent, dacă acesta este 0 atunci x^0 este 1 și trebuie sa afișăm doar coeficientul, dar și aici trebuie să verificăm dacă acel coeficient este în primul rând diferit de 0 și, în al doilea rând, dacă este pozitiv respectiv negativ. Se continuă apoi și pentru cazul în care exponentul este 1 sau dacă are oricare altă valoare, de fiecare dată verificându-se și coeficientul și returnând mereu un alt mod de afișare.

Exemplu pentru cazul în care face verificarea pentru exponent = 0:



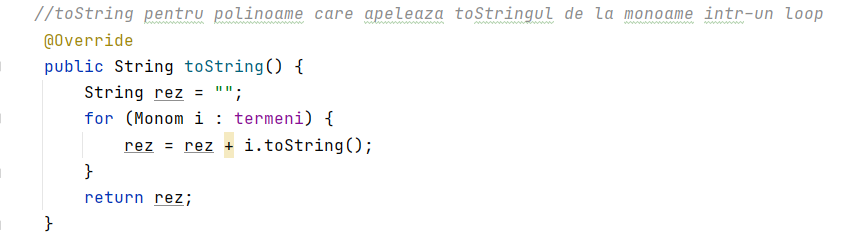
Tot în clasa monom am creat și doua metode pentru adunare, respectiv pentru scădere pe polinoame, metode ce urmează să fie apelate în metodele din clasa polinom.



**Clasa polinom**

Clasa polinom primește o listă de monoame numită termeni, iar pe lângă constructori și override la metoda toString din monom, are toate metodele pentru operații.

În metoda toString am parcurs lista de monoame și am adunat fiecare monom găsit la rezultat, apelând la fiecare loop toString-ul pentru monoame.



**Metoda pentru adunare și metoda pentru scădere**

În primul rând, am creat polinomul rezultat și variabile pentru numărul de termeni din fiecare polinom, precum și monoame auxiliare. La început am făcut o verificare pentru că apărea un caz în care dacă pun de exemplu 2x^0 în primul polinom, iar în al doilea nu am nici un monom cu exponentul 0, nu mai punea 2 la rezultatul final, și a trebui sa verific acest caz.

În al doilea rând, am făcut într-un while pentru verificarea ”cât timp mai avem monoame” apoi am luat primul monom din primul polinom respectiv primul monom din al doilea polinom și le-am comparat exponenții, se adaugă la rezultatul final mereu exponentul mai mare, iar dacă se găsesc două monoame cu același exponent se apelează funcția de adunare din monom și punem suma lor în lista de monoame finala, apoi se trece la următoarele două polinoame. Am procedat la fel și pentru metoda de scădere, dar am adăugat o înmulțire cu -1 pentru atunci când avem termeni cu + în al doilea polinom, iar minusul din față trebuie sa le schimbe semnul.

**Metoda pentru înmulțire**

Pentru înmulțire am parcurs monoamele din primul polinom și din al doilea polinom în paralel și de fiecare data am aplicat formula specifică înmulțirii: (coeficient \* coeficient) x^ (exponent + exponent) și am pus rezultatul în lista de monoame de la polinomul rezultat. Am adăugat și o parte în care pun toate monoamele rezultate într-o lista de monoame auxiliare și o parcurg pentru a găsi monoamele ce au exponenți egali în rezultat și îi adun între ei astfel încât sa nu apară mai mulți exponenți cu aceeași putere la rezultatul final. Apoi mult toate monoamele din lista auxiliara în lista de rezultate.

**Metoda pentru derivare și metoda pentru integrare**

În cazul acestor metode se folosește câte un singur polinom, aici este considerat ca fiind primul polinom introdus. În ambele operații se parcurg pe rând monoamele din polinom și atâta timp cât exponentul monomului este mai mare decât zero se aplică formula de derivare, respectiv formula de integrare. Pentru derivare coeficientul se înmulțește cu exponentul, iar la exponent se scade 1 din el (coef \* exp) x^ (exp - 1). Pentru integrare a trebuit sa transform coeficientul în float, inițial declarat ca și int, astfel încât atunci când facem coeficient împărțit la exponent să avem un rezultat mai precis. Formula de la integrare folosită este (coef / (exp + 1)) x^ (exp + 1).

**Clasa ModelCalc**

În clasa ModelCalc am făcut metoda regex, metodă care verifică polinomul introdus de utilizator dacă respectă un anumit pattern și anume ”([-+]?\\d\*)[xX]\\^(\\d?)”.

Am folosit clase din pachetul regex, în obiectul de tip Pattern am pus string-ul declarat ca fiind modelul pe care polinomul trebuie sa îl respecte, iar în obiectul de tip matcher am pus polinomul care este dat ca și parametru la metodă. Pe urmă, cât timp este găsit un matcher care să respecte pattern-ul dat, se pune coeficientul și exponentul separat în int-uri după ce au fost convertite din string în int folosind funcția parseInt din clasa Integer. Cele doua int-uri, putere și coeficient, le punem într-un obiect de clasa Monom și îl adăugăm în polinomul final.

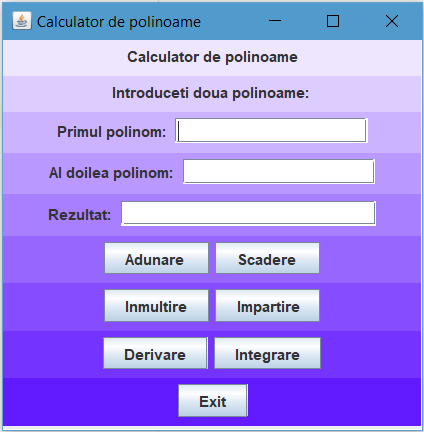
**Pachetul view**

**Clasa ViewCalc**

Clasa ViewCalc este clasa aparte din MVC în care se creează interfața cu care va lucra utilizatorul. Este modul prin care se poate oferi o comandă și se primește răspuns, fără ca utilizatorul sa știe cum a fost generat acel răspuns, fără să știe partea din spate a aplicației.

Clasa are ca și atribut principal lucrul cu elementele din pachetul swing și awt și moștenirea clasei JFrame. La început au fost declarate obiectele ce urmează să fie folosite pe parcurs precum label-uri, textfield-uri și butoane. După, bazându-mă și pe elementele din model am creat un constructor care îmi creează practic toată interfața. Am urmat ideea de a împărți panel-ul principal în mai multe panel-uri mai mici, primul panel are titlul, al doilea comanda, al treilea, al patrulea și al cincilea conțin label-uri și textfield-uri pentru primul polinom, al doilea polinom, respectiv pentru rezultat. Pe urmă am grupat butoanele cu operații câte două pe panel ( panel-urile 6, 7, 8), iar în panel-ul al nouălea am pus butonul de exit. Toate cele nouă panel-uri le-am pus pe panel-ul principal, am dat un titlu întregii aplicații, și am setat mărimile ferestrei (350/350). Tot în această clasă am declarat pentru fiecare buton câte un listener și getter-e și setter-e pentru polinoame, respectiv rezultat.

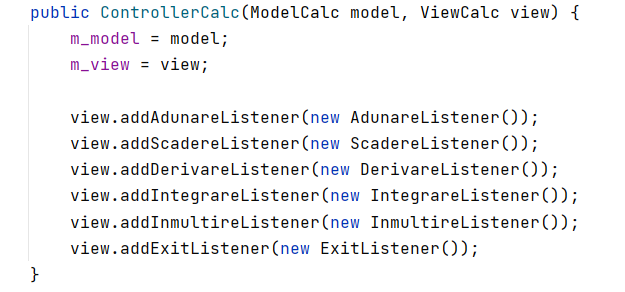
Finalul interfeței grafice arată așa:



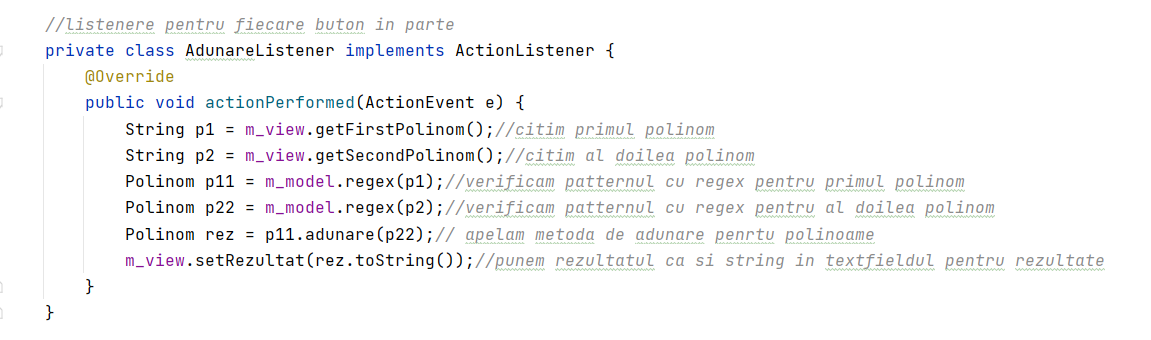
**Pachetul controller**

**Clasa ControllerCalc**

Clasa ControllerCalc, cea care face legătura între model și view, conține în principal un constructor și listener pentru fiecare buton. Constructorul este important pentru că în main se va crea un obiect final care să reprezinte controllerul și care trebuie sa conțină atât view cat și model pentru a le lega. Pe lângă parametri de view și model apar și obiectele ce fac referință la butoanele cu listener declarate în view:

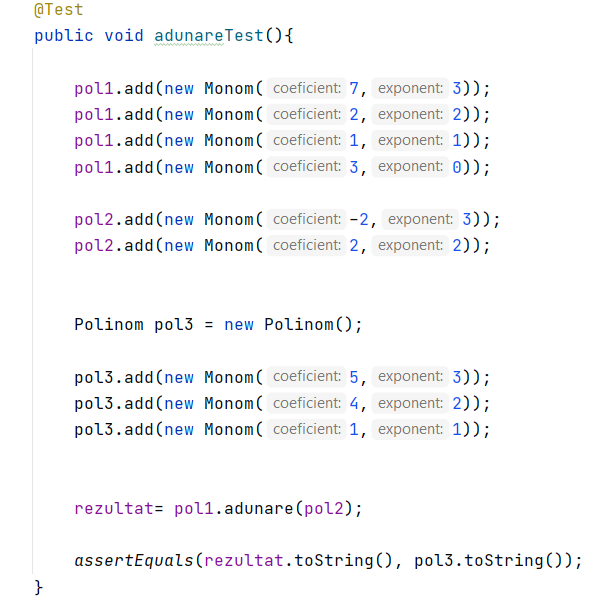


În cazul acțiunilor de la buton se respectă aproximativ aceiași pași ca în exemplu de mai jos, unde apare ActionListener pentru butonul de adunare. Pașii sunt: citirea celor doua polinoame introduse de utilizator, verificarea dacă respectă pattern-ul prin metoda regex din model, butonul funcționează doar dacă este introdus un polinom corect, altfel nu face nimic, după ce au fost verificați polinoamele se aduna, apelând metoda adunare din polinom și se pune rezultatul în polinomul rez, la final se setează textfield-ul care arată rezultatul ca fiind rez transformat în string, desigur, folosind metoda toString din polinom.

****

1. **Rezultate**

Pentru testare am folosit JUnit și am creat clasa PolinomTest. În această clasa am făcut câte o metoda de test pentru fiecare operație. Am creat manual Monoame cu coeficient și exponent pe care le am adăugat în listele de monoame, adică în cele două polinoame. Am creat un al treilea polinom care reprezintă rezultatul care ar trebui să fie returnat în urma operației aplicate asupra celor doua polinoame date. Folosind funcția assertEquals am testat dacă rezultatul obținut este egal cu al treilea polinom introdus. Am procedat întocmai și pentru celelalte operații, cu câte un test pentru fiecare. În exemplul de mai jos este testul pentru adunarea polinoamelor.



1. **Concluzii**

Per total, întregul proiect este destinat utilizării, și datorită testărilor și verificărilor făcut aș putea spune că ar trebui să funcționeze pentru orice caz pe care utilizatorul îl introduce. Singura operație pe care nu am reușit să o implementez este împărțirea și aș putea să o adaug pe lista de lucruri ce pot fi adăugate. O altă dezvoltare ulterioară pe care aș putea să o adaug este modul în care este afișat rezultatul, pentru ca aș vrea să scot acel plus de la început, precum și formatul coeficienților care dacă nu au zecimale după virgulă să fie afișați fără .0.

Ce am reușit eu să deprind din acest proiect este modul prin care se utilizează metodele de tip regex și cum putem să verificăm dacă un text introdus de noi se potrivește cu criteriile cerute, lucru care am observat că este folosit în mai multe domenii precum introducerea unui email, sau a unui cod numeric personal și așa mai departe. Pe lângă aceasta, am reușit să învăț cum funcționează JUnit-ul și cum putem folosi un proiect de tip Meaven.

Fiind primul proiect, am început să mă familiarizez și cu GitLab și încărcarea proiectului pe această platformă. În ceea ce privește codul, pot spune că am fost deja familiarizată cu structura proiectului MVC din anul anterior, dar am învățat cum să lucrez mai ordonat și mai logic într-un astfel de proiect.

1. **Bibliografie**
2. LucidChart pentru USE CASE Diagram

<https://lucid.app/lucidchart/>

<https://youtu.be/zid-MVo7M-E>

1. Regex

<https://regex101.com/r/vOfwT8/1>

<https://youtu.be/4XZdpwchIt4>

<https://crunchify.com/what-is-regex-pattern-regular-expression-how-to-use-it-in-java-example-attached/>

1. UML Diagram

<https://www.jetbrains.com/help/idea/class-diagram.html>

1. JUnit tutorial

<https://www.baeldung.com/junit-5?fbclid=IwAR39Om7OF7lZzhNzm3-3cNbMOJmgNlu7bA5uD5aQXXejnh7PDhtamxzjJoU>

1. Operații cu polinoame

<https://www.mathsisfun.com/algebra/polynomials-sums-products-roots.html>

<https://www.mathsisfun.com/algebra/polynomials-multiplying.html>