DOCUMENTAȚIE

TEMA 1

Nume Student: IRINI KARINA

Grupa: 30229

CUPRINS:

1. Obiectivul temei

Obiectivul principal al temei este acela de proiectare și implementare al unui calculator de polinoame cu o interfață grafică dedicată prin care utilizatorul poate introduce polinoame, selecta operația matematică de efectuat și vizualiza rezultatul.

Pentru a îndeplini obiectivul principal al temei, trebuie să ne axăm și pe obiectivele secundare. Este important să începem cu analiza problemei și identificarea cerințelor. Această etapă ne ajută să ne concentrăm asupra funcționalităților de bază ale calculatorului de polinoame, aspect detaliat în capitolul următor. Proiectarea aplicației, descrisă în cel de-al treilea capitol, este esențială pentru a începe implementarea programului. Aceasta constă în realizarea următoarelor operații matematice: adunarea, scăderea, înmulțirea, împărțirea, derivarea și integrarea polinoamelor, precum și a unei intefețe grafice. Este necesară realizarea acesteia într-un mod intuitiv, pentru a asigura o utilizare cât mai plăcută a software-ului. Ultimul pas, dezvoltat în capitolul cinci, asigură buna funcționare a calculatorului, fiind reprezentat de testarea codului creat.

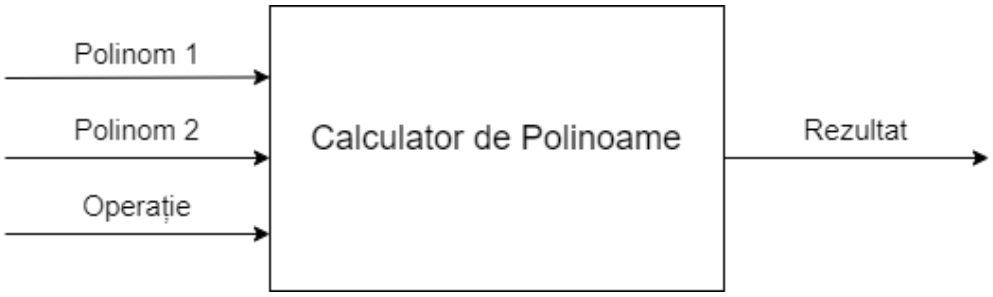
2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Cerințele funcționale sunt cele care descriu funcționalitățile pe care programul trebuie să le îndeplinească, iar cerințele non funcționale sunt cele care descriu caracteristicile programului.

Cerințele problemei subliniază dezvoltarea unei interfețe grafice prin care utilizatorul să aibă posibilitatea de a introduce două polinoame, de a selecta operația matematică dorită care poate fi aleasă din următoarele șase posibilități: adunare, scădere, înmulțire, împărțire, derivare sau integrare. Aplicația trebuie să folosească algoritmi potriviți, astfel încât toate aceste operații să fie realizate cu succes, iar utilizatorul să poată observa rezultatul generat sub o formă cât mai clară.

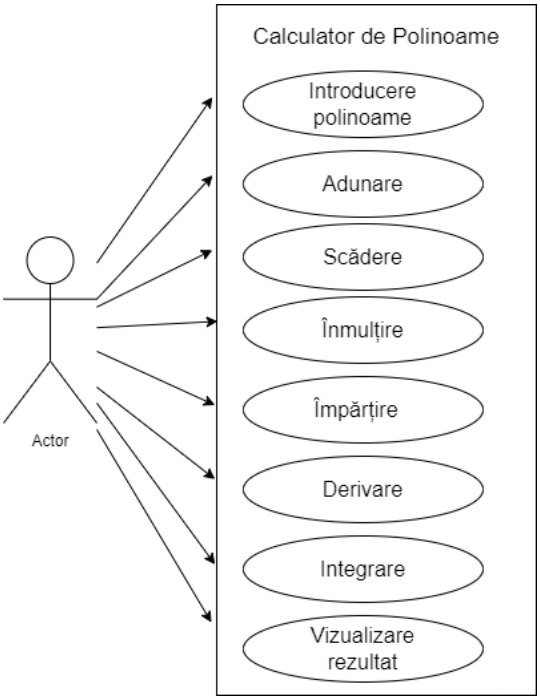
Una dintre cerințele non funcționale esențiale constă în utilizarea cât mai ușoară și intuitivă a programului. Astfel, utilizatorul nu ar trebui să reușească să introducă polinoame invalide sau să nu selecteze operația dorită. În cazul acestor scenarii, el va fi avertizat de greșeala făcută, prin mesaje sugestive. O altă cerință non funcțională este reprezentată de fiabilitatea programului, siguranța acestuia față de posibilele erori în timpul rulării.

Pentru a înțelege mai bine modelarea calculatorului de polinoame, am folosit următoarea schemă:



Calculatorul are o singură ieșire, rezultatul, și trei intrări, reprezentate de cele două polinoame și operația, în cazul adunării, scăderii, înmulțirii și împărțirii, în cazul derivării și integrării fiind nevoie doar de un polinom și de selecția operației.

Pentru a descrie funcționalitatea unei aplicații, este importantă evidențierea diferitelor cazuri de utilizare ale acesteia. Pentru calculatorul de polinoame, funcționalitatea se bazează pe operațiile implementate, care trebuie să ofere utilizatorului un rezultat final satisfăcător.



Cazurile de utilizare sunt similare pentru toate operațiile matematice, având ca și actor principal utilizatorul. Ca și exemplu voi prezenta scenariul pentru împărțirea polinoamelor.

Scenariul principal de succes:

1. Utilizatorul introduce 2 polinoame în casetele text corespunzătoare din interfața grafică.
2. Utilizatorul selectează operația de împărțire.
3. Utilizatorul apasă butonul de efectuare a operației selectate.
4. Calculatorul de polinoame realizează împărțirea celor două date de intrare și afișează rezultatul, format din cât și rest.

Scenarii alternative:

- Polinoamele sunt incorecte.
    - Utilizatorul introduce incorect polinoamele (ex. Cu mai multe caractere de semn “-+x^4” sau pune caracterul care sugerează urmarea unei puteri, dar fără a o introduce “3x^”).
    - Calculatorul de polinoame semnalează aspectul prezentat anterior.
    - Scenariul se întoarce în pasul 1.

- Se încearcă împărțirea la zero.
    - Utilizatorul introduce al doilea polinom ca fiind 0.
    - Calculatorul de polinoame semnalează că nu poate efectua operația.
    - Scenariul se întoarce în pasul 1.
- Se încearcă împărțirea unui polinom cu un grad mai mic la un polinom cu grad mai mare. Acest scenariu urmează aceiași pași cu cel prezentat anterior.
- Nu se selectează operația.
    - Utilizatorul introduce polinoamele.
    - Utilizatorul apasă butonul de efectuare, dar fără a selecta operația.
    - Calculatorul de polinoame semnalează acest aspect.
    - Scenariul se întoarce în pasul 2.
- Se încearcă realizarea operației fără introducerea ambelor polinoame.
    - Utilizatorul introduce doar un polinom.
    - Utilizatorul selectează operația dorită și apasă butonul de efectuare.
    - Calculatorul de polinoame semnalează lipsa celuilalt polinom.
    - Scenariul se întoarce în pasul 1.

3. Proiectare

Programarea orientată pe obiecte (OOP) este o paradigmă de programare ce se bazează pe conceptul de obiecte, componente software care incorporează atât atributele cât și operațiile care se pot efectua asupra câmpurilor. Unul dintre principiile fundamentale ale OOP este încapsularea datelor, ascunderea detaliilor unui obiect și, astfel, protejarea informațiilor. Prin utilizarea principiilor OOP se beneficiază de o dezvoltare mai rapidă și mai eficientă a software-ului dorit, cu o structură clară și organizare bună a datelor. Acest aspect implică identificarea obiectelor și a interacțiunilor dintre acestea, definirea claselor și a relațiilor dintre ele, implementarea metodelor și a datelor pentru fiecare clasă, precum și testarea codului.

Astfel, pentru această tema am ales să utilizez două clase principale pentru definirea obiectelor:

1. Clasa Monomial: Un polinom este format din unul sau mai multe monoame. Monoamele sunt definite prin două atribute: putere (degree) care este întotdeauna un număr întreg, și coeficient (coefficient) care poate stoca numere cu zecimale, aspect esențial în cazul operațiilor de împărțire și integrare. De asemenea, această clasă utilizează și clasele Pattern și Matcher pentru a realiza transformarea unui șir de caractere într-un polinom corespunzător.
2. Clasa Polynomial: Pentru a defini un polinom este nevoie de o colecție de monoame. Pentru a stoca aceste date într-un mod eficient, am utilizat clasa TreeMap din biblioteca standard Java. Această clasă oferă o colecție sortată de perechi cheie-valoare, implementată printr-un arbore binar de căutare, ceea ce permite accesul rapid la datele stocate și permite căutarea, inserarea și ștergerea elementelor în timp logaritmic. În cazul nostru, cheia este reprezentată de puterea monomului, aceasta fiind unică, iar valoarea de monomul însuși.

Proiectul este de asemenea împărțit în mai multe pachete:

1. Pachetul Graphical User Interface (gui): Acest pachet conține clasele care implementează interfața grafică – View, Controller. Clasa View, care extinde clasa JFrame din biblioteca standard Swing, este responsabilă pentru toate interferențele vizuale, pe când clasa Controller, care implementează interfața ActionListener, este responsabilă pentru gestionarea interacțiunilor utilizatorului și a datelor de intrare și ieșire în cadrul aplicației.
2. Pachetul Data Models (model): Acest pachet conține clasele care modelează datele aplicației – Polynomial, Monomial.
3. Pachetul Business Logic (logic): Acest pachet conține clasa care implementează funcționalitățile operațiilor matematice, metodele pentru adunarea, scăderea, înmulțirea și împărțirea a două polinoame, precum și derivarea, integrarea unui polinom – Operations. Această clasă folosește structura de date ArrayList pentru a returna rezultatele operației de împărțire.

Din diagrama UML de pachete se poate observa că între pachetele Graphical User Interface, respectiv pachetul Business Logic și pachetul Data Model există o relație de dependență , analog între pachetul Business Logic și pachetul Data Models.

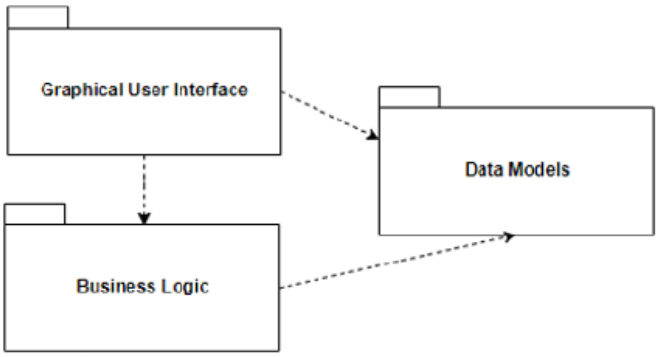
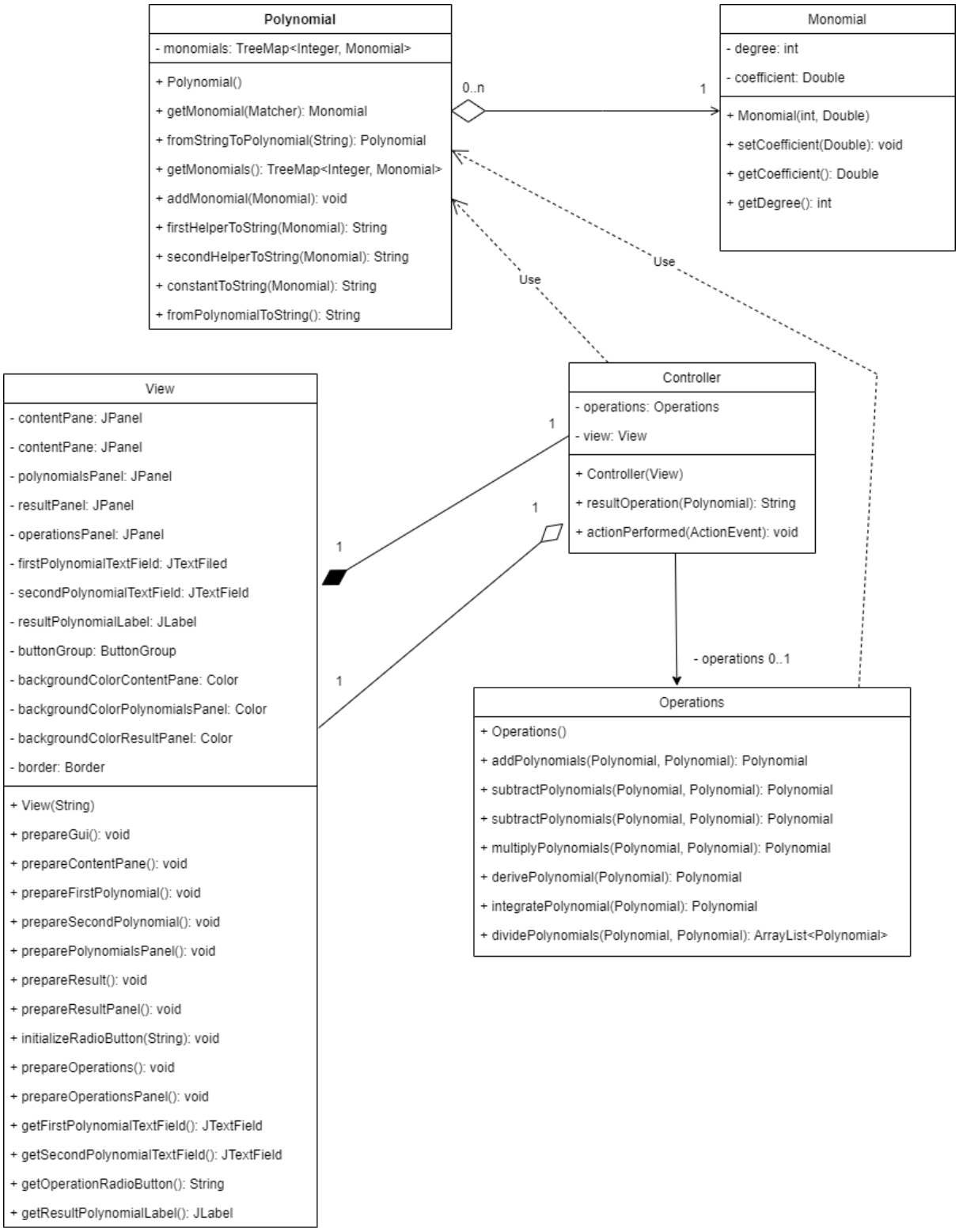


Diagrama UML de clase, din care se poate observa relațiile dintre acestea, este următoarea:



4. Implementare

Clasa Monomial: Clasa are drept câmpuri puterea (degree), care este întotdeauna un număr întreg, și coeficientul (coefficient) fiecărui monom, care poate stoca numere cu zecimale, aspect esențial în cazul operațiilor de împărțire și integrare. Pentru a putea manipula aceste atribute ca niște obiecte și pentru a le putea atașa metode, am ales să le declar sub forma claselor învelitoare aferente: Integer și Double. Folosind încapsularea datelor, a fost necesară implementarea setterelor și getterelor, împreună cu un constructor corespunzător.

Clasa Polynomial: În această clasă există un câmp numit "monomials", folosit pentru stocarea monoamelor care alcătuiesc polinomul. Am ales să utilizez o structură de date TreeMap, unde cheia este reprezentată de puterea monomului, garantând astfel unicitatea acestei, iar valoarea de monomul însuși. Pentru a facilita implementarea operațiilor, în special operația de împărțire, am ales să stochez monoamele în ordine descrescătoare a puterii lor. Analog clasei Monomial, a fost necesară implementarea unui getter pentru accesarea câmpului.

Pentru a adăuga un monom într-un polinom, am creat o metodă care, înainte de a introduce monomul furnizat ca parametru în structura TreeMap, verifică dacă există deja alte monoame cu același grad. În caz afirmativ, se setează un nou coeficient ca fiind suma celor doi.

Pentru a transforma un șir de caractere într-un polinom corespunzător, am creat două metode care îi oferă comfort utilizatorul în momentul introducerii polinoamelor dorite și îl avertizează în cazul în care nu a furnizat un polinom valid. Pentru a realiza această conversie am furnizat un tipar cât mai potrivit metodei compile din cadrul clasei Pattern, din pachetul Regex care se ocupă de expresiile regulate. Am ales să folosesc un cod relativ vast și în cazul transformării unui polinom într-un șir de caractere pentru a mă apropia cât mai mult posibil de un calculator de polinoame real și potrivit pentru orice tip de persoană care accesează această aplicație.

Clasa Operations: Această clasă este destinată funcționalităților operațiilor matematice. Fiecare dintre acestea este realizată printr-o metodă separată.

Metodele addPolynomials(Polynomial polynomial1, Polynomial polynomial2) și subtractPolynomials(Polynomial polynomial1, Polynomial polynomial2) se ocupă de adunarea, respectiv scăderea a două polinoame furnizate ca argumente și returnează polinomul rezultat. Aceaste metode iterează prin primul polinom folosind o buclă foreach, iar pentru fiecare monom, iterează și prin al doilea, asigurând succes în cazul oricărui scenariu:

- În cazul în care toate monoamele celor două polinoame au aceeași putere, se adună, respectiv se scad coeficienții corespunzători, apoi se creează un monom cu gradul comun și noul coeficient, care este adăugat în polinomul rezultat.
- Unele monoame dintr-un polinom nu au corespondent de exponent în celălalt. Aici, pe lângă însumarea sau scăderea coeficienților, monoamele care nu îndeplinesc acest criteriu sunt adăugate în rezultat, în cazul operației de scădere, cu semn schimbat.
- Monoamele celor două polinoame nu au nicio putere comună, deci se introduc monoamele fiecărui polinom în rezultat.

Metoda multiplyPolynomials(Polynomial polynomial1, Polynomial polynomial2) se ocupă de înmulțirea a două polinoame furnizate ca argumente și returnează un polinom rezultat numit resultMultiplication. Această metodă parcurge fiecare polinom, înmulțind monom cu monom, ceea ce înseamnă adăugarea în polinomul rezultat de monoame cu coeficienții înmulțiți și puterile adunate.

Metoda derivePolynomial(Polynomial polynomial) se ocupă de derivarea polinomului furnizat ca argument și returnează un polinom rezultat numit resultDerivative. Această metodă parcurge monoamele polinomului și adaugă în polinomul rezultat un nou monom care are ca și coeficient produsul dintre coeficientul monomului curent și puterea sa, iar ca exponent, gradul minus 1.

Metoda integratePolynomial(Polynomial polynomial) se ocupă de integrarea polinomului furnizat ca argument și returnează un polinom rezultat numit resultIntegration. Această metodă parcurge monoamele polinomului și adaugă în polinomul rezultat un nou monom care are ca și coeficient coeficientul monomului curent împărțit la puterea sa, iar ca exponent, gradul plus 1. Constanta aferentă integrării este adăugată în momentul afișării rezultatului în interfața grafică.

Metoda dividePolynomials(Polynomial polynomial1, Polynomial polynomial2) se ocupă de împărțirea polinoamelor furnizate ca argumente și, în cazul în care se poate efectua operația, returnează o structură de tip ArrayList de obiecte Polynomial care reprezintă câtul și restul. Algoritmul se bazează pe următorii pași:

1. Împărțirea polinomului cu cel mai mare grad la celălalt polinom cu grad mai mic (presupunem că polynomial1 are cel mai mare grad)
2. Împărțirea primului monom al lui polynomial1 la primul monom al lui polynomial2 și obținerea primului termen al câtului
3. Înmulțirea câtului cu polynomial2 și scăderea înmulțirii din polynomial1, obținând restul împărțirii
4. Repetarea procedurii începând cu pasul 2 considerând restul ca noul deîmpărțit al împărțirii, până când gradul restului este mai mic decât cel al lui polynomial2

În cazul în care se încearcă împărțirea unui polinom cu grad mai mic la un polinom cu grad mai mare sau împărțirea la 0, rezultatul va fi null, ceea ce va genera ca în TextField-ul aferent rezultatului din interfața grafică să se afișeze mesajul de eroare “Cannot do this operation!”.

Clasa View se ocupă de interfața grafică și de toate interacțiunile vizuale. Pentru o organizare bună a codului, au fost create mai multe metode. Fiecare dintre acestea ajută la pregătirea componentelor. Interfața este alcătuită din două TextField-uri însoțite de două Label-uri cu texte sugestive: “First Polynomial”, “Second Polynomial” care permit introducerea de către utilizator a celor două polinoame dorite. Acesta poate selecta operația dorită prin intermediul a șase RadioButton-uri cu nume semnificative și poate vizualiza rezultatul în dreptul Label-ului “Result” prin apăsarea singurului buton existent, care are asociat un ActionListener. În situația în care utilizatorul încearcă afișarea rezultatului, dar fără a selecta operația dorită sau fără a introduce cele două polinoame, acesta va fi avertizat de greșelile realizate cu ajutorul unor dialoguri informative sau de eroare.

Clasa Controller, care implementează interfața ActionListener, are rolul de a coordona interacțiunea între clasa View și cele din pachetul model, respectiv logic. Metoda actionPerformed preia selecția în privința operației matematice selectate alături de cele două polinoame introduse, efectuează operația corespunzătoare și cu ajutorul metodei resultOperation afișează rezultatul. Dacă utilizatorul nu furnizează polinoame valide, acesta va fi notificat de mesajul “Invalid polynomial input!” afișat în cea de-a treia casetă text.

5. Rezultate

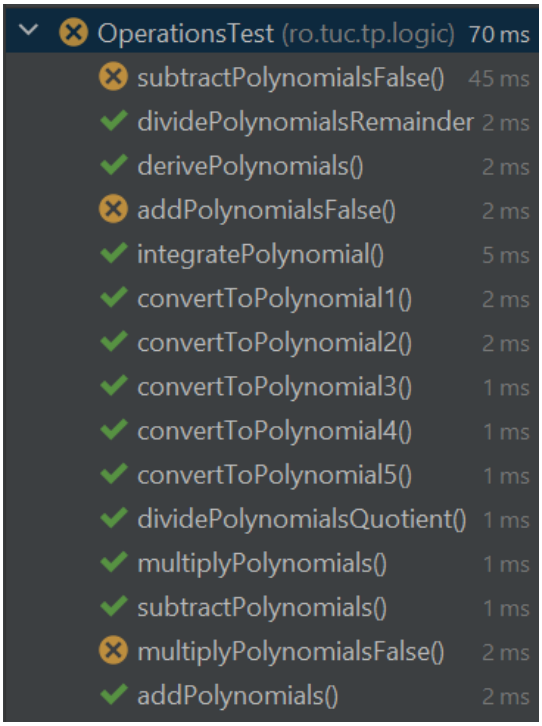
Pentru a reduce riscul de erori și de comportament imprevizibil al programului, am utilizat framework-ul JUnit de testare unitară pentru limbajul de programare Java. Testarea unitară este un proces de testare a fiecărei unități de cod individuală pentru a se asigura că aceasta produce rezultatele așteptate.

Clasa în care se găsesc testele, a fost creată în pachetul test al proiectului de tip Maven, tip care oferă o execuție automatizată a procesului de testare. Astfel, pentru verificarea operației de adunare, m-am asigurat că suma polinoamelor 3x^2-x și x-2 convertită într-un șir de caractere este aceiași cu "3x^2-2". Cu ajutorul aserțiunilor am verificat comportamentul a două dintre scenariile metodei de adunare, precum și a metodei care convertește un polinom într-un șir de caractere potrivit. Analog, m-am asigurat că și celelalte metode care implementează funcționalitățile operațiilor matematice au un rezultat conform așteptărilor. În acest sens, am creat și câteva teste "false", care știam că vor eșua, aspect care s-a și întâmplat.

De asemenea, a fost necesară și testarea metodelor pentru interfața grafică de citire a unui șir de caractere și transformarea acestuia într-un polinom valid, astfel încât utilizatorul să beneficieze de o experiență cât mai eficientă și comodă. În acest caz, am testat cât mai multe cazuri posibile, pentru a mă asigura că se semnalează o introducere invalidă a polinoamelor dorite.

```
@Test
void addPolynomials(){
    assertEquals(Controller.resultOperation(operations.addPolynomials(polynomial1, polynomial2)), actual: "3x^2-2");}
```

```
@Test
void convertToPolynomial4() { assertNull(Polynomial.fromStringToPolynomial( polynomialString: "NotAPolynomial")); }
```



După ce programul a trecut toate testele implementate, m-am convins că aplicația funcționează corect și că obiectivele au fost atinse cu succes.

6. Concluzii

Această tema m-a ajutat să percep importanța etapei de implementare a unui proiect. Înainte de a începe partea de proiectare și scriere de cod pentru o aplicație, este esențial să ne gândim la ce dorim să obținem și, mai ales, în ce fel vrem să lucrăm, ce structuri de date ne sunt de folos, ce scenarii am putea să gestionăm, precum și alte detalii importante care trebuie puse la punct. Timpul meu a fost, astfel, mult mai bine organizat, știind anterior cum este nevoie să acționez pentru un rezultat cât mai satisfăcător.

Întâlnirea conceptelor noi, precum lucrul cu expresii regulate și modelul arhitectural Model-View-Controller (MVC) a fost de asemenea de folos. Împărțirea codului în cele trei părți ale MVC m-a ajutat să dobândesc abilitatea de a-mi structura mult mai bine codul și de a mă disciplina în privința organizării. Faptul că am învățat să utilizez interfața Regex și clasele Pattern, Matcher pentru a găsi un șablon pentru împărțirea unui șir de caractere în fragmentele dorite, îmi va folosi și în viitor, fiind o metodă mult mai ușoară de implementat și de înțeles, decât cele cunoscute anterior.

O posibilă dezvoltare ulterioară ar fi capabilitatea calculatorului de a prelucra coeficienți zecimali sau puteri raționale. Acest aspect ar mări spectrul de utilizare al aplicației. De asemenea, ar putea să gestioneze mai multe operații matematice, cum ar fi integrarea definită sau calculul limitelor polinoamelor. Aceste operații pot fi ușor implementate, cu un nivel mediu de cunoștințe în domeniul matematicii.

7. Bibliografie

1. https://dsrl.eu/courses/pt/
2. https://regex101.com/r/nI5oA5/6
3. https://stackoverflow.com/
4. https://users.utcluj.ro/~igiosan/teaching_poo.html
5. https://app.diagrams.net/
6. https://www.tutorialspoint.com/java/java_regular_expressions.htm