-Tehnici de programare-

Tema 1: Calculator de polinoame

Student: Iamnițchi Bogdan

Grupa: 30225

Cuprins:

1. Obiectivul temei

1.1 Obiectivul principal:

Tema are ca obiectiv principal proiectarea și implementarea unei aplicații “calculator de polinoame “. Datele de intrare vor fi introduse de către utilizator printr-o interfață utilizator, datele vor fi procesate și prelucrate în funcție de operația pe care utilizatorul o va alege, rezultatul fiind afișat tot în interfațautilizator.

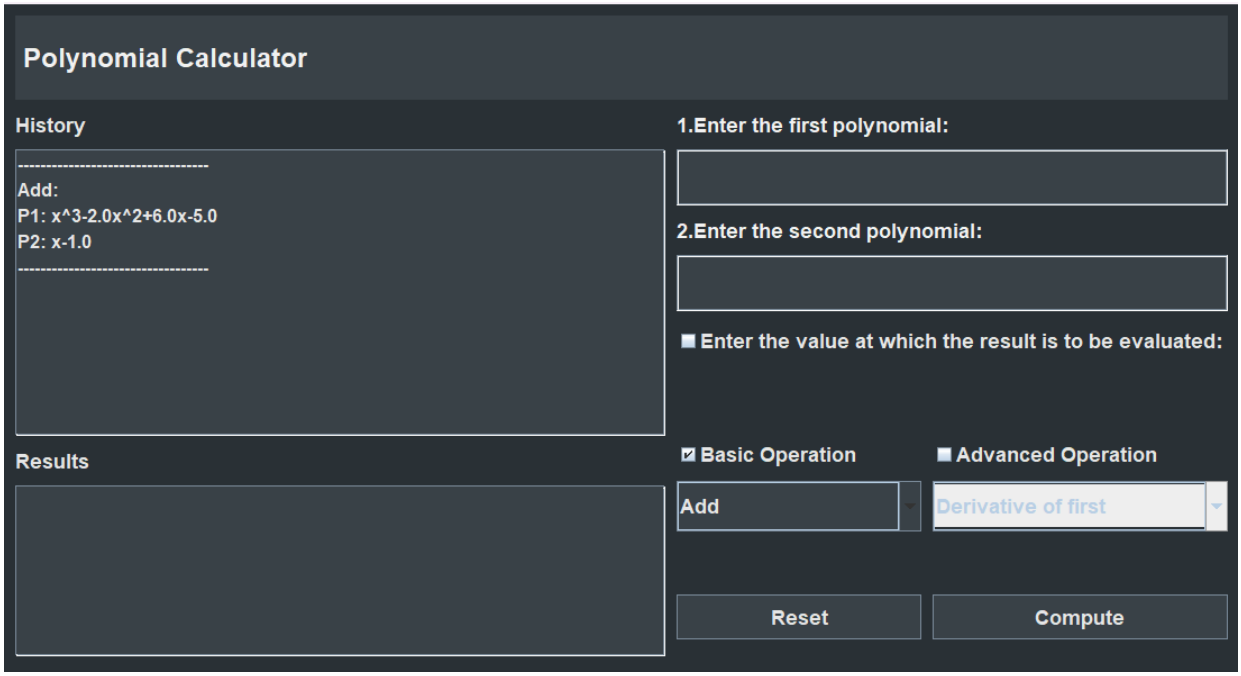
1.2 Obiective secundare:

a. Dezvoltarea de use case-uri și scenarii - posibile cazuri de funcționare a calculatorului
b. Alegerea structurilor de date - abstractizarea datelor
c. Împărțirea pe clase - modelarea problemei
d. Dezvoltarea algoritmilor - implementarea algoritmilor cu ajutorul operațiilor matematice
e. Implementarea soluției - proiectarea ideii
f. Testare - JUnit Test

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

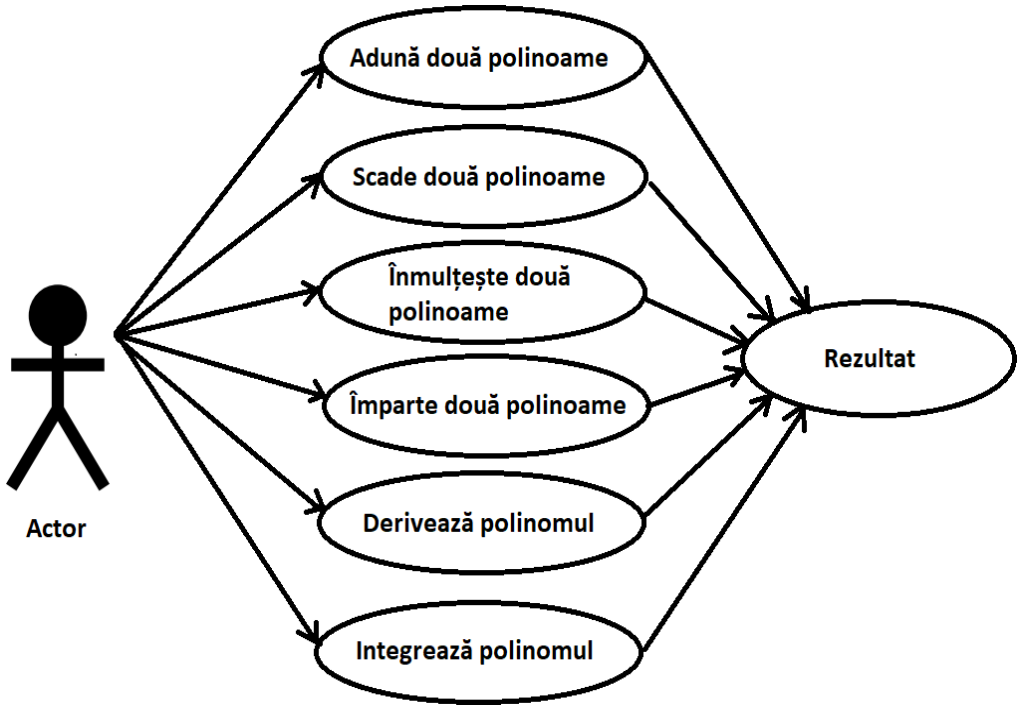
Aplicația este reprezentată sub o formă grafică, reprezentând interfața cu utilizatorul. Printre funcționalitățile de bază ale acesteia se numără: deschiderea / închiderea aplicației și realizarea operațiilor cerute de utilizator.

Aceasta aplicație este modelată în așa fel încât să îndeplinească toate cerințele funcționale ale unui calculator de polinoame. Ca mod de organizare a informației este folosită încapsularea, accentul fiind pus pe abstractizrea datelor care împreună cu proprietățile și caracteristicile lor sunt grupate în obiecte, reunite mai apoi în clase.



Utilizatorul poate introduce în primele două text field-uri (1. Și 2.) polinoame de forma: ax^y+bx^z+cx^1+dx^0, ax^y+bx^z+cx+d, ax^y+bx+cx^y+d+ex sau alte forme mixte, oricare dintre acestea fiind considerate validă. După introducerea polinoamelor are loc transformarea acestora din String în obiecte polinom pe care se pot face operații. Această transformare se face separând fiecare termen al polinomului în funcție de caracterul "+" (unde era – se înlocuiește cu +- pentru a putea avea loc această separare). Mai departe se tratează 10 cazuri, fiecare dintre ele reprezentând un mod de scriere a unui monom ca String, iar în funcție de fiecare caz are loc convertirea acestuia pentru adaugarea corecta in HashMap, cu cheie (exponent) și valoare (coeficient).

După introducerea polinomului sau polinoamelor asupra căruia/cărora se va efectua operația, ultizatorul alege care dintre operații vrea să o execute. La apăsarea oricarui buton din cele 6, operația aleasă va fi efectuată și rezultatul va fi afișat în text field-ul etichetat cu Q (exceptie face operația de împărțire, unde rezultatul este dat sub formă de cât și rest; câtul va fi afișat în text field-ul etichetat cu Q și restul în text field-ul etichetat cu R).



USECASE : ***Operatia de adunare***

ACTOR : Un utilizator care dorește să aplice operația de adunare asupra a două polinoame.

PRE-CONDITIONS: Cele două polinoame(datele de intrare care urmeaza sa fie prelucrate) pe care utilizatorul dorește să aplice operația de adunare au fost introduse în câmpurile specifice din interfața utilizator. Datele de intrare sunt în conformitate cu formatul acceptat de către calculatorul curent (cel prezentat mai sus).

POST-CONDITIONS: Rezultatul adunării celor două polinoame de intrare va fi afișat în câmpul de rezultat al interfeței programului.

NORMAL FLOW: Utilizatorul introduce datele intrare în campurile puse la dispozitie pentru fiecare polinom, solicită operația de adunare prin selectarea optiunii “aduna”, iar datele vor fi preluate, prelucrate și furnizate în câmpul de “rezultat” al interfeței cu utilizatorul sub forma de String.

USECASE : ***Operatia de scădere***

ACTOR : Un utilizator care dorește să aplice operația de scădere asupra a două polinoame.

PRE-CONDITIONS: Cele două polinoame pe care utilizatorul dorește să aplice operația de adunare au fost introduse în câmpurile specifice din interfața utilizator. Datele de intrare sunt în conformitate cu formatul acceptat de către calculatorul curent.

POST-CONDITIONS: Rezultatul scăderii celor două polinoame de intrare va fi afișat în câmpul de rezultat al interfeței programului.

NORMAL FLOW: Utilizatorul introduce datele intrare în campurile puse la dispozitie pentru fiecare polinom, solicită operația de scădere prin selectarea optiunii de "scade", iar datele vor fi preluate, prelucrate și furnizate în câmpul de "rezultat" al interfeței cu utilizatorul sub forma de String.

USECASE : ***Operatia de înmulțire***

ACTOR : Un utilizator care dorește să aplice operația de înmulțire asupra a două polinoame.

PRE-CONDITIONS: Cele două polinoame pe care utilizatorul dorește să aplice operația de adunare au fost introduse în câmpurile specifice din interfața utilizator. Datele de intrare sunt în conformitate cu formatul acceptat de către calculatorul curent.

POST-CONDITIONS: Rezultatul înmulțirii celor două polinoame de intrare va fi afișat în câmpul de rezultat al interfeței programului.

NORMAL FLOW: Utilizatorul introduce datele intrare în campurile puse la dispozitie pentru fiecare polinom, solicită operația de înmulțire prin selectarea optiunii "inmulteste", iar datele vor fi preluate, prelucrate și furnizate în câmpul de "rezultat" al interfeței cu utilizatorul sub forma de String.

USECASE : ***Operatia de împărțire***

ACTOR : Un utilizator care dorește să aplice operația de împărțire asupra a două polinoame.

PRE-CONDITIONS: Cele două polinoame pe care utilizatorul dorește să aplice operația de adunare au fost introduse în câmpurile specifice din interfața utilizator.

Datele de intrare sunt în conformitate cu formatul acceptat de către calculatorul curent.
POST-CONDITIONS: Rezultatul împărțirii celor două polinoame de intrare va fi afișat în cele două câmpuri rezultat ale interfeței programului (câtul va fi afișat în text field-ul etichetat cu Q și restul în text field-ul etichetat cu R).
NORMAL FLOW: Utilizatorul introduce datele intrare în campurile puse la dispozitie pentru fiecare polinom, solicită operația de împărțire prin selectarea optiunii "imparte", iar datele vor fi preluate, prelucrate și furnizate în câmpurile de "rezultat" ale interfeței cu utilizatorul sub forma de String.

USECASE: ***Operatia de derivare***
ACTOR: Un utilizator care dorește să efectueze operația de derivare pentru un polinom.
PRE-CONDITIONS: Polinomul pe care dorim sa il derivăm a fost introdus in câmpul corespunzător din interfața programului (operația se face pentru un singur polinom, citit din primul text field-A). Datele de intrare sunt în conformitate cu formatul acceptat de către calculatorul curent.
POST-CONDITIONS: Dacă primul text field este gol, operația nu se va putea efectua, asfel se va genera un mesaj de eroare în câmpul etichetat cu Q care ne va transmite să introducem polinomul în primul text field-A. Dacă polinomul a fost introdus în câmpul corespunzător și datele sunt valide, atunci rezultatul operației va fi afișat în câmpul rezultat.
NORMAL FLOW: Utilizatorul introduce datele de intrare în primul câmp pus la dispoziție pentru polinom, solicită operația de derivare prin selectarea optiunii "deriveaza", iar datele vor fi preluate și furnizate în câmpul rezultat al interfetei sub forma de string.

USECASE: ***Operatia de integrare***
ACTOR: Un utilizator care dorește să efectueze operația de integrare pentru un polinom.
PRE-CONDITIONS: Polinomul pe care dorim sa îl integram a fost introdus in câmpul corespunzător din interfața programului (operația se face pentru un singur polinom, citit din primul text field-A). Datele de intrare sunt în conformitate cu formatul acceptat de către calculatorul curent.

POST-CONDITIONS: Dacă primul text field este gol, operația nu se va putea efectua, asftel se va genera un mesaj de eroare în câmpul etichetat cu Q care ne va transmite să introducem polinomul în primul text field-A. Dacă polinomul a fost introdus în câmpul corespunzător și datele sunt valide, atunci rezultatul operației va fi afișat în câmpul rezultat.
NORMAL FLOW: Utilizatorul introduce datele de intrare în primul câmp pus la dispoziție pentru polinom, solicită operația de integrare prin selectarea optiunii "integreaza", iar datele vor fi preluate și furnizate în câmpul rezultat al interfetei sub forma de string.

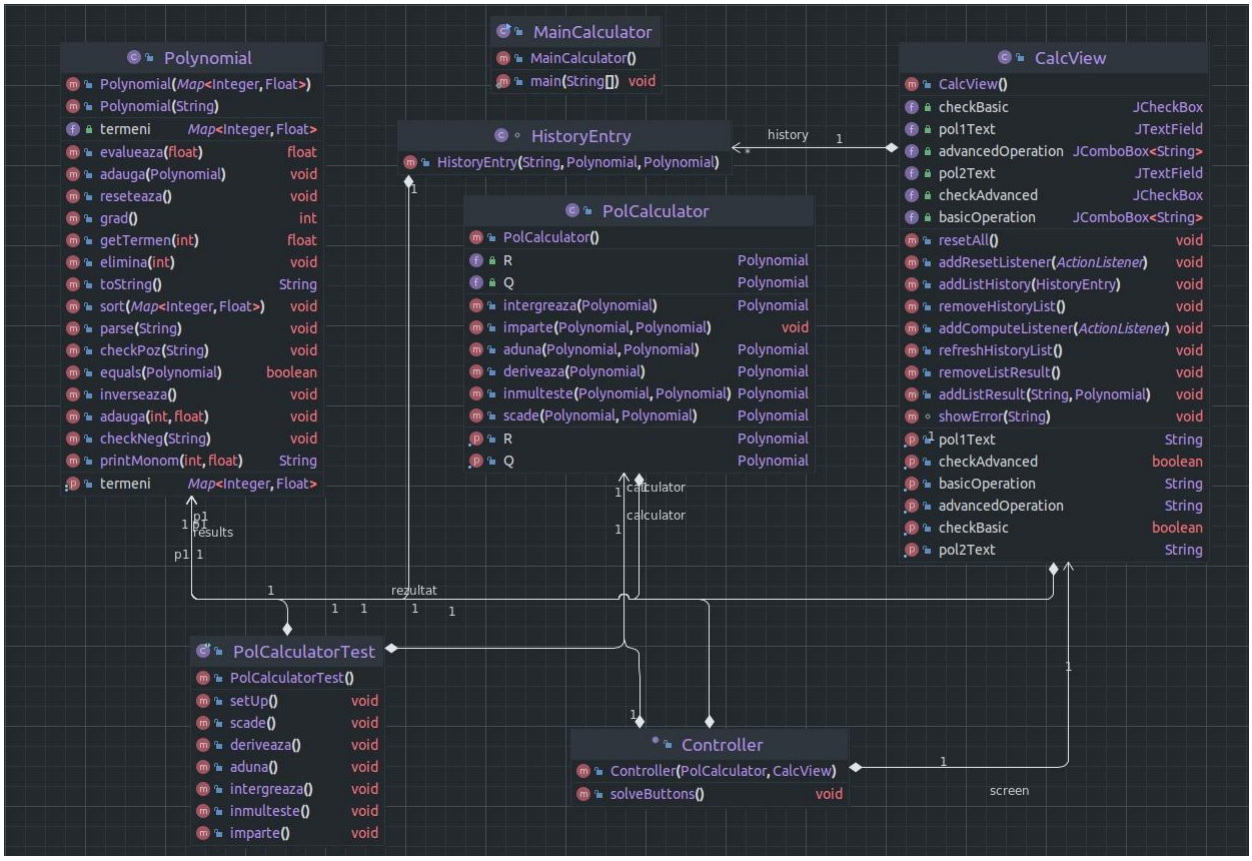
3. Proiectare

În ceea ce priveste proiectarea aplicației am optat pentru folosirea a 5 clase, bine-delimitate, atfel incat sa fie mult mai clara si mult mai simpla ideea de implemenatre.
Prima clasă cu care am inceput a fost clasa Polynomial, care urma sa aiba o variabila instanta de tip String si un HashMap de termeni asociat, pentru initializarea acestora folosindu-se un constructor. Stiind ca polinomul va fi primit ca String printr-o interfata utilizator, am realizat ca trebuie sa fac transformarea polinomului dat ca String intr-un HashMa, dar si o metoda care sa transforme obiectul de tip polinom in obiect de tip String care sa fie mai apoi afisat pe ecran.
Avand clasa Polynomial, am incercat sa gandesc proiectarea urmatoarei clase, PolCalculator, ca fiind responsabila cu implementarea operatiilor: adunare, scadere, inmultire, impartire, derivare, integrare. Pentru fiecare operatie am gandit un mic algoritm dupa care l-am implementat in cod. In functie de necesitatile fiecarei operatii am mai facut ajustari in clasa Polynomial.
In clasa CalcView, urma a fi implementata interfata cu utilizatorul, aceasta clasa avand rol strict de View. In ceea ce priveste aspectul grafic mi-am facut o mica schema pe foaie pe care am transpus-o mai tarziu pe calculator.
Am implementat dupa si o clasa de controller numita Controller, care face legatura dintre CalcView (GUI) si PolCalulator respectiv Polynomial. Controller-ul proceseaza datele introduse si adauga ascultatori pentru butoane. Functionalitatea acestei interfete se datoreaza clasei PolClaculator care are rolul de Model, furnizand logica dupa care se ghideaza problema.

3.1. Structuri de date

Pentru a ma obisnui sa lucrez cu structuri de date abstracte am folosit un HashMap pentru a retine un polinom. Am ales această structură de date deoarece în comparație cu un vector normal, HashMap are deja implementate metodele de put(), remove(), size(), get(), (de care mă pot folosi atunci cand doresc să construiesc un polinom), iar pentru un vector aceste metode ar trebui definite, implementate și mai apoi folosite. Totodată gestionarea memoriei cu aceasta structura de date este foarte eficienta.

3.2. Diagrama de clase UML



4. IMPLEMENTARE

Clasa **Polynomial**, are un constructor cu parametru de tip String. Acest String urmeaza a fi parsat in functie de semnul +, pentru aceasta se foloseste functia split. Dupa aceasta impartire rezulta un tablou de stringuri, pe fiecare pozitie aflandu-se un monom. Avand acest tablou de stringuri, vom face transformarea fiecarui element din tablou intr-un entry (exponent, coeficient), care va fi adaugat apoi într-un HashMapde termeni apartinand unui polinom. Pentru a vedea in ce fel trebuie transformat stringul in monom, am facut testarea pe 10 cazuri: 5 pozitive (se apeleaza cu metoda checkPoz) si 5 negative (se apeleaza cu metoda checkNeg). Aceasta clasa mai contine 2 gettere (getTermen(), gettermeni()), 1 settere(setTermeni()), toString() – care converteste polinomul in obiect de tip String, cu ajutorul functiei printMonom; adauga() - adauga un entry (exponent, coeficient) in hashMap; grad() - calculeaza si returneaza gradul polinomului; inverseaza() – schimba semnul fiecarui termen din polinom, prin inmultirea coeficientului cu -1ș sort() - metodă care are rolul de a sorta termenii unui polinom în ordinea descrescătoare a exponentului; evalueaza() -metoda care evalueaza un polinom cu o valoare data; equals() – returneaza daca polinomul this este la fel ca un alt polinom dat.

Clasa **PolCalculator**, este clasa responsabila de efectuarea operatiilor, avand totodata rolul de Model (în arhitectura MVC) furninzand operatiile si modul de functionare. Acesta clasa contine urmatoarele metode:

ADUNARE:
Această metodă are rolul de a aduna două polinoame primite ca parametru și returnează un alt polinom care reprezintă suma acestora. Ideea de bază a algoritmului este următoarea: se parcurge fiecare element din primul polinom si fiecare element din al doilea pana cand unul din polinoame se termină, adunand termenii care au exponentul egal sau copiind în polinomul rezultat termenul care are exponentul mai mare decât în celalalt polinom.

SCADERE:
Această metodă are rolul de a scadea două polinoame (pol1 si pol2) primite ca parametru și returnează un alt polinom care reprezintă diferenta acestora. Pentru realizarea operatiei de scadere ne folosim de doua metode mentionate anterior, anume inverseaza si adunare. Pe scurt, facem o adunare intre pol1 si pol2 cu semnele inversate, ceea ce este echivalent cu o scadere.

INMULTIRE:
Această metodă are rolul de a inmulti două polinoame (pol1 si pol2) primite ca parametru și returnează un alt polinom care reprezintă rezulatul acestei operatii. Inmultirea se efectueaza element cu element, creeând cate un monom pentru fiecare inmultire dintre 2 termeni, pe care il adaugam apoi la polinomul rezultat.

IMPARTIRE:
Această metodă are rolul de a imparti două polinoame (P si B) primite ca parametru și construieste 2 polinoame reprezentand catul si restul impartiriii celor doua polinoame. Pentru implementarea acestei operatii se urmeaza un algoritm care are in vedere: sortarea si restrangerea polinoamelor primate ca parametru, atribuirea polinomului P lui R(polinomul de rest), dupa care se executa niste instructiuni in mod repetat cat timp gradul lui R este mai mare sau egal cu gradul lui B(gradul este dat de cel mai mare exponent al polinomului). Instructiunile care se vor repeta cat timp conditia este adevarata sunt: crearea unui monom care are coeficientul egal cu (coeficientul primului element din R / coeficientul primului element din B) iar exponentul este egal cu (coeficientul primului element din R - coeficientul primului element din B), lui P i se atribuie R, se creaza un nou polinom N in care vom pune monomul construit anterior, acelasi monomul adaugam si polinomului Q (polinomul care reprezinta catul), inmultim polinomul N cu B, iar rezultatul il scadem din P, in urma scaderii avand rezultatul in R.

DERIVARE:
Această metodă are rolul de a deriva un polinom primit ca parametru. Pentru implementarea acestei operatii se parcurge polinomul dat ca parametru si se construieste un alt polinom ale carui monoame au coeficientul egal cu coeficientul monomului din polinomului initial inmultit cu exponentul acestuia, iar exponentul este egal cu exponentul monomului din polinomului initial-1.

INTEGRARE:
Această metodă are rolul de a integra un polinom primit ca parametru. Pentru implementarea acestei operatii se parcurge polinomul dat ca parametru si se construieste un alt polinom ale carui monoame au coeficientul egal cu (coeficientul monomului din polinomului initial / exponentul acestuia+1) iar exponentul este egal cu exponentul monomului din polinomului initial+1.

In clasa **CalcView** sunt declarate toate obiectele ce fac posibila implementarea interfetei cu utilizatorul: text field uri, etichete, butoane, panoul care este definit folosind Java Swing Design Form. Pentru fiecare buton este implementat un ActionListener unde se descrie functionalitatea pe care trebuie sa o aiba. Pentru oricare din butoane are loc citirea stringului din text field/-uri si crearea polinomului/polinoamelor cu care se va efectua operatia urmand ca rezultatul sa fie afisat in unul sau in ambele text field uri rezultat.

Clasa **Controller**, face lagatura dintre **PolCalculator** care implementeaza operatiile si **CalcView**, care reprezinta interfata cu utilizatorul. Aceasta clasa se asigura ca datele sunt corecte dupa accea le da mai departe pentru a fi rezolvate dar si este responsabila de actiunea butoanelor deoarece de aici definim si adaugam ascultatori pt butoane folosind functii ajuatoare din **CalcView**.

Clasa **MainClass**, joaca rolul de Executabil din aceasta clasa se creaza obiectele ce reprezinta Model, View si Controller in functia main.

5. REZULTATE

Pentru observarea rezultatelor și a comportamentului operatiilor am ales sa folosesc o clasa de testare cu JUnit. Initial am definit polinoamele de care voi avea nevoie, cat si “calculatorul”. Am construit “calculatorul” definit mai sus, deoarece vom avea nevoie de acesta pe tot parcursul executarii metodelor din clasa de testare. In schimb, constructorul pentru fiecare polinom a fost apelat in metoda setUp(), urmand sa fie create in fiecare dintre metodele de test.

```
@BeforeEach
public void setUp() {
    try {
        p1 = new Polynomial( sursa: "x^3-2x^2+6x-5");
        p2 = new Polynomial( sursa: "x^2-1");
    }
    catch (Exception e) {
        System.out.println("Invalid Polynomial testing...");
    }
}
```

Am ales sa fac cate o metoda de test pentru fiecare operatie: adunare, scadere, inmultire, impartire, derivare si integrare, fiecare dintre metode avand numele operatiei precedat de “test” (ex: testAdunare, testDerivare), iar inainte de fiecare metoda am pus “@Test” pentru a arata faptul ca in acea metoda se testeaza ceva.

Voi prezenta doua scenarii de testare, restul fiind asemanatoare cu cele prezentate:

1. Primul scenariu de testare este pentru operatia de adunare. In metoda testAdunare() se creeaza doua polinoame, dupa care in polinomul rezultat punem suma acestora, urmand ca sa facem verificarea cu assertTrue care testeaza daca parametrul primit este true sau false. Aceasta functie returneaza true in caz de egalitate si false in caz ca acestea nu corespunde. In final se afiseaza un mesaj, care ne spune ca operatia s-a efectuat cu success. Tot testul este într-un bloc try-catch deoarece adunarea polinoamelor poate duce la aruncarea unei execepti in cazul in care polinomul nu a fost introdus corect de la tastatura.

```
@Test
public void aduna() {
    try {
        result = calculator.aduna(p1, p2);
        assertTrue(result.equals(new Polynomial( sursa: "x^3-x^2+6x-6")));
    }
    catch (Exception e) {
        System.out.println("Something went wrong...");
    }
    finally {
        System.out.println("Test passed successfully");
    }
}
```

2. Al doilea scenariu de testare este asemanator cu primul, diferenta este ca aici se foloseste doar un polinom pentru realizarea operatiei, nu doua ca si la adunare.

```java
@Test
public void deriveaza() {
    try {
        result = calculator.deriveaza(p1);
        assertTrue(result.equals(new Polynomial( sursa: "3x^2-4x+6")));
    }
    catch (Exception e) {
        System.out.println("Something went wrong...");
    }
    finally {
        System.out.println("Test passed successfully");
    }
}
```

6. CONCLUZII

Aceasta tema m-a ajutat sa-mi dezvolt abilitatile de lucru in ceea ce priveste programarea orientate pe obiecte, lucrul cu InteliJ (fiind primul meu proiect realizat in acest mediu de dezvoltare) si nu numai.

Am invatat sa proiectez o interfata cu utilizatorul cat mai practica si usor de folosit, am invatat sa folosesc debbuger-ul si sa-mi corectez codul, sa-l fac cat mai lizibil. Totodata, am exersat modul de lucru cu structuri de date de tip HashMap si mi-am dat seama mai bine ce inseamna procesul de abstractizare a datelor.

7. BIBLIOGRFAFIE

- https://www.jetbrains.com/
- http://coned.utcluj.ro/~marcel99/PT2020/
- https://mkyong.com/tutorials/junit-tutorials/
- https://www.codecademy.com/learn/learn-java
- https://www.geeksforgeeks.org/