DOCUMENTATIE

TEMA 1

Calculator de polinoame

NUME STUDENT: Jișa Diana-Maria
GRUPA: 30224

CUPRINS

1. Obiectivul temei

Obiectivul acestei teme de laborator este de a propune, proiecta si implementa un calculator de polinoame, cu o singură variabilă și cu coeficienți întregi. Acesta trebuie să ofere posibilitatea efectuării de operații precum adunarea, scăderea, înmulțirea, împărțirea, derivarea și integrarea polinoamelor. Sistemul de calcul va avea în plus, o interfață grafică prietenoasă pentru utilizator, care să faciliteze accesul și utilizarea într-un mod cât mai intuitiv și eficient. Scopul final al acestei teme este de a oferi utilizatorului rezultate precise și clare în urma efectuării calculelor.

Obiective secundare:

- Analiza problemei și identificarea cerințelor funcționale și non-funcționale ale calculatorului de polinoame (Cap. 2)
- Proiectarea soluției utilizând paradigma OOP, diagrame UML de clase și pachete, și definirea structurilor de date și a interfețelor necesare. (Cap. 3)
- Implementarea sistemului de calcul și a interfeței grafice utilizator (Cap. 4)
- Realizarea testelor unitare utilizând utilitarul JUnit și prezentarea scenariilor de testare (Cap. 5)
- Concluzii și sugestii pentru îmbunătățiri ulterioare (Cap. 6)
- Referințe bibliografice (Cap. 7)

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

Un polinom este o expresie algebrică formată din mai multe monoame, fiecare având un coeficient și un exponent pozitiv. În cadrul calculatorului pe care l-am implementat, polinomul este introdus de utilizator sub forma unui șir de caractere (String), care este transmis constructorului. Pentru a separa monoamele și a prelua coeficienții și exponenții acestora, am utilizat Pattern și Matcher în constructor, conform cerințelor, dar cu toate acestea metoda implementata de mine nu este cea mai eficientă.

După apariția interfeței grafice pe ecran și introducerea de la tastatură a celor două polinoame, utilizatorul trebuie să apese unul dintre cele 6 butoane pentru a alege operația dorită:

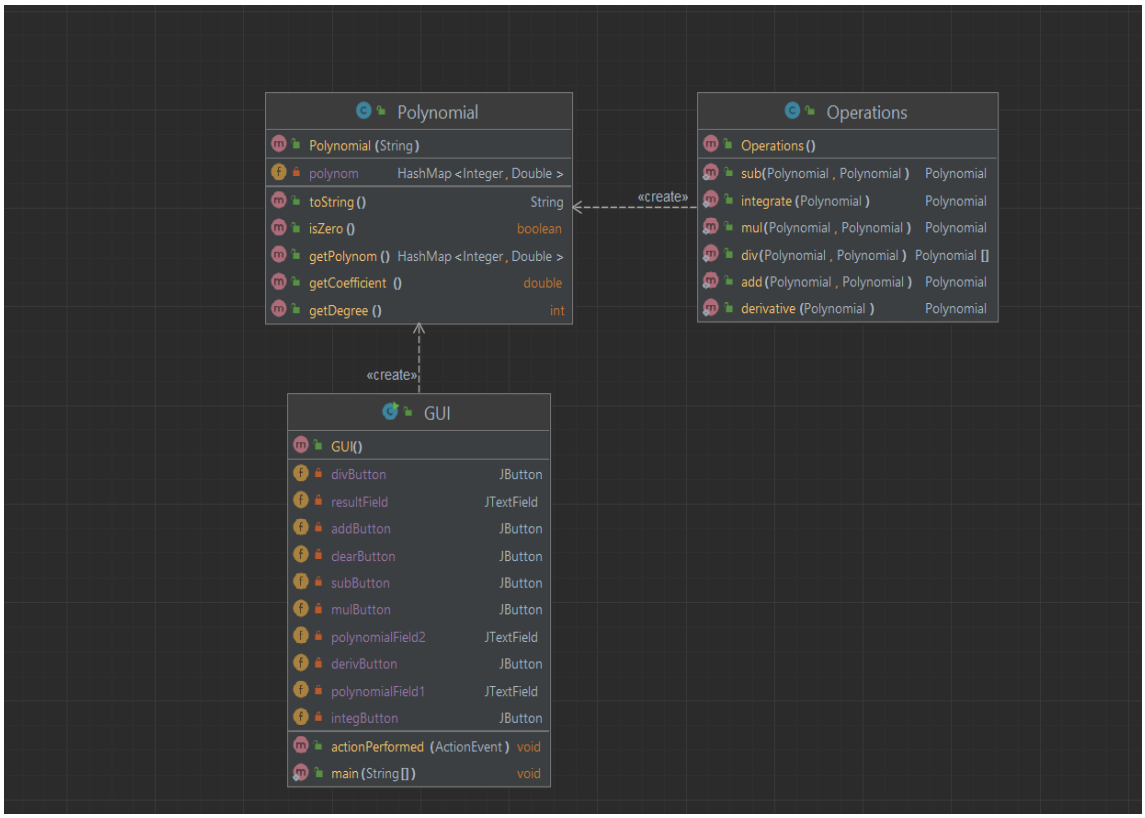
- Adunarea a două polinoame
- Scăderea a două polinoame
- Înmulțirea a doua polinoame
- Înpărțirea a doua polinoame
- Derivarea unui polinom
- Integrarea unui polinom

Toate cele 6 operații implementate oferă un rezultat corect. Pentru operațiile de derivare si integrare este de ajuns dacă se completează doar câmpul corespunzător primului polinom. Dupa apăsarea unui buton, rezultatul operației selectate va fi afișat în câmpul corespunzător, iar la apăsarea unui alt buton, rezultatul se va actualiza automat. (Exemplu de polinom valid: -x^3+2x^2-5x+4). Interfața calculatorului mai include și un buton Clear, care poate fi apăsat pentru a șterge polinoamele introduse de utilizator și pentru a reseta rezultatul.

Este important ca utilizatorul să introducă un polinom care conține doar variabila x, altfel va fi aruncată o excepție. De asemenea, nu este permisă separarea unui monom și a unui operator prin spațiu, altfel se va arunca, de asemenea, o excepție. Este necesar să simplificăm polinomul pentru ca calculatorul să funcționeze corect. Acest lucru înseamnă că nu putem scrie polinomul sub forma x^2+2x^2, deoarece aceasta nu este o formă simplificată. În schimb, ar trebui să scriem direct 3x^2, care este forma simplificată a acestui polinom.

3. Proiectare

Diagrama UML



Clasa "Operations" utilizează clasa "Polynomial" ca tip de date pentru parametrii și rezultatele metodelor sale, dar nu există o relație de moștenire sau de compunere între cele două clase. În schimb, clasa "GUI" interacționează cu ambele clase, "Polynomial" și "Operations", fiind responsabilă de interacțiunea utilizatorului cu interfața grafică a aplicației.

Pentru a organiza modulele și pentru a grupa clasele și interfețele asociate, se folosesc pachete Java. În acest caz, fiecare dintre cele trei clase implementate este plasată într-un pachet separat. Astfel, clasa "Polynomial" se află în pachetul "DataModels", clasa "Operations" se află în pachetul "BusinessLogic", iar clasa "GUI" se află în pachetul "GraphicalUserInterface".

În ceea ce privește interacțiunea cu structurile de date, am folosit în principal obiecte de tipul **HashMap<Integer, Double>** pentru reprezentarea unui polinom. În această mapă, cheile sunt puterile polinomului, iar valorile sunt coeficienții. Am utilizat o mapă deoarece putem accesa coeficienții în funcție de puterea corespunzătoare în mod eficient.

4. Implementare

Pentru implemntarea acestui calculator de polinoame, am utilizat 3 clase: **Operations, Polynomial** si **GUI**, plus inca o clasa **PolynomialTest** pentru testarea cu utilitarul Junit.

Operations

Aceasta este o clasă Java din pachetul "org.example.BussinesLogic", care conține metodele pentru efectuarea operațiilor cu polinoame.

- Metoda **add(p1, p2)** primește două polinoame, p1 și p2, ca parametri și returnează un nou polinom care reprezintă suma lor. În implementarea acestei metode, mai întâi se creează un nou obiect Polynomial gol, numit result, pentru a stoca rezultatul. Apoi, se iterază prin toți termenii primului polinom și se adaugă fiecare termen la result, adunând coeficienții termenilor care au aceeași putere. În final, se adaugă la result orice termeni din p2 care nu apar în p1.
- Metoda **sub(p1, p2)** primește două polinoame, p1 și p2, ca parametri și returnează un nou polinom care reprezintă diferența lor. Această metodă este similară cu add(p1, p2), cu excepția faptului că coeficienții termenilor din p2 sunt scăzuți din coeficienții termenilor din p1, în loc să fie adunați.

```
public static Polynomial sub(Polynomial p1, Polynomial p2) {
    Polynomial result = new Polynomial("");
    p1.getPolynom().forEach((power, coef) -> {
        double dif = coef - p2.getPolynom().getOrDefault(power, defaultValue: 0.0);
        result.getPolynom().put(power, dif);
    });
    p2.getPolynom().forEach((power, coef) -> {
        if (!result.getPolynom().containsKey(power)) {
            double diff = -coef;
            result.getPolynom().put(power, diff);
        }
    });
    return result;
}
```

- Metoda **mul(p1, p2)** primește două polinoame, p1 și p2, ca parametri și returnează un nou polinom care reprezintă produsul lor. În implementarea acestei metode, se iterază prin toți termenii ambelor polinoame și se înmulțesc coeficienții termenilor care aparțin acelorași puteri. Produsele astfel obținute sunt adăugate la un nou obiect Polynomial, numit result, la puterea care reprezintă suma puterilor termenilor.

- Metoda **div(p1, p2)** primește două polinoame, p1 și p2, ca parametri și returnează un array de două polinoame - reprezentând câtul și restul împărțirii. În implementarea acestei metode, mai întâi se verifică dacă p2 este zero, caz în care se aruncă o excepție. Apoi, se creează două noi obiecte Polynomial - quotient și remainder, inițializate cu p1 și respectiv cu 0. Cât timp gradul remainder-ului este mai mare sau egal cu gradul p2-ului, se adaugă la polinomul quotient un nou termen format din coeficientul obținut prin împărțirea coeficienților termenilor cu aceeași putere, și din puterea care reprezintă diferența dintre gradele remainder-ului și p2-ului. Acest nou termen este apoi înmulțit cu p2 și scăzut din remainder. Procesul se repetă până când remainder-ul are un grad mai mic decât p2 sau devine zero.

```
public static Polynomial[] div(Polynomial p1, Polynomial p2) {
    if (p2.isZero()) {
        throw new IllegalArgumentException("Cannot divide by zero.");
    }

    Polynomial quotient = new Polynomial("");
    Polynomial remainder = p1;
    while (!remainder.getPolynom().isEmpty() && remainder.getDegree() >= p2.getDegree()) {
        int PowerDif = remainder.getDegree() - p2.getDegree();
        double CoefRatio = remainder.getCoefficient() / p2.getCoefficient();
        Polynomial term = new Polynomial("");
        term.getPolynom().put(PowerDif, CoefRatio);
        quotient = Operations.add(quotient, term);
        Polynomial product = Operations.mul(p2, term);
        remainder = Operations.sub(remainder, product);
        if(remainder.getCoefficient()==0)
            break;
    }
    return new Polynomial[]{quotient, remainder};
}
```

- Metoda **derivative(p)** calculează derivata unui polinom dat. Se folosește de formula de derivare a unui polinom, unde coeficienții termenilor sunt multiplicați cu puterile corespunzătoare și apoi se scade 1 din putere. Pentru termenul liber (puterea 0), derivata este 0.
- Metoda **integrate(p)** calculează integrala unui polinom dat. Se folosește de formula de integrare a unui polinom, unde coeficienții termenilor sunt împărțiți la puterile corespunzătoare plus 1.

```
public static Polynomial integrate(Polynomial p) {
    Polynomial result = new Polynomial("");
    p.getPolynom().keySet().forEach((power) -> {
        double coef = p.getPolynom().get(power) / (power + 1);
        result.getPolynom().put(power + 1, coef);
    });
    result.getPolynom().put(0, 0.0);
    return result;
}
```

Polynomial

Această clasă din pachetul "org.example.DataModels" conține un polinom stocat sub forma unei colecții de tip HashMap<Integer, Double>, unde cheia reprezintă puterea termenilor polinomului, iar valoarea asociată cheii reprezintă coeficientul termenului respectiv. Clasa dispune și de metodele "**getPolynom**", "**getDegree**" și "**getCoefficient**" care permit accesul la colecție, dar și la gradul și coeficientul unui termen specific al polinomului. Metoda "**isZero**" este utilizată pentru a verifica dacă polinomul este zero, prin verificarea faptului că toți coeficienții sunt egali cu zero.

Tot în această clasă este suprascrisă si metoda "**toString**", care afișează polinomul sub formă de șir de caractere, în ordinea descrescătoare a puterilor variabilei x, astfel încât coeficienții mai mari sunt afișați mai întâi. În plus, această metoda adaugă semnele de plus și minus în mod corespunzător și omitw termenii cu coeficienții zero.

Cea mai importantă parte din aceasta clasă o reprezintă constructorul **public Polynomial (String polynomial)** care primește un șir de caractere ce reprezintă un polinom și construiește obiectul Polynomial corespunzător. Constructorul creează un obiect HashMap (o colecție de perechi cheie-valoare), numit "polynom", care va fi utilizat pentru a stoca coeficienții și puterile termenilor din polinomul dat. Expresia regulată definită în variabila "termPattern" este utilizată pentru a găsi monomii polinomului din șirul dat. Această expresie regulată permite găsirea termenilor care conțin numere, litere (reprezentând variabila x) și exponenți, și să se identifice semnul termenului, fie +, fie -. Conform șablonului, constructorul ia termenul găsit, îl analizează pentru a determina puterea și coeficientul și îl adaugă la HashMap.

```java
public Polynomial(String polynomial) {
    polynom = new HashMap<>();
    Pattern termPattern = Pattern.compile( regex: "[+-]?[\\d\\.]*[a-zA-Z]?\\^?\\d*");
    Matcher termMatcher = termPattern.matcher(polynomial);

    while (termMatcher.find()) {
        String term = termMatcher.group();
        if (term.isEmpty())
        { continue; }
        if (term.equals("x")) {
            polynom.put(1, 1.0);
        } else if (term.contains("x^")) {
            String[] parts = term.split( regex: "x\\^");
            double coef = 1;
            int power = Integer.parseInt(parts[1]);
            if (parts.length > 0 && !parts[0].isEmpty()) {
                if(parts[0].equals("-"))  coef=-1;
                else if(parts[0].equals("+")) coef=1;
                else coef=Double.parseDouble(parts[0]);
            }
            polynom.put(power, coef);
        } else if (term.contains("x")) {
            double coef;
            int power = 1;
            if (term.startsWith("-x")) coef = -1;
            else if (term.startsWith("+x")) coef = 1;
            else  coef = Double.parseDouble(term.replace( target: "x",  replacement: ""));
            polynom.put(power, coef);
        } else {
            double coef = Double.parseDouble(term);
            polynom.put(0, coef);
        }
```

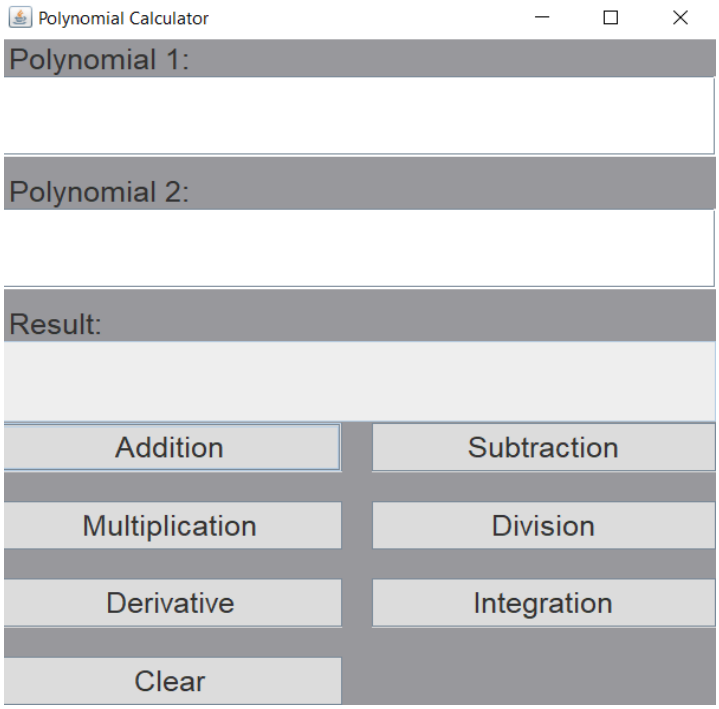
GUI

Această clasă reprezintă o interfață grafică de utilizator (GUI) pentru un calculator de polinoame. Ea este definită în cadrul pachetului org.example.GraphicalUserInterface și implementează interfața ActionListener. Clasa conține câmpuri pentru introducerea polinoamelor, afișarea rezultatelor, butoane pentru diferite operații (adunare, scădere, înmulțire, împărțire, derivare, integrare și ștergere), precum și metode pentru tratarea evenimentelor asociate apăsării acestor butoane.

Pentru a construi GUI-ul, începem prin setarea unui layout de tip BorderLayout și crearea unui panou dedicat pentru polinoame. Acest panou conține trei câmpuri pentru introducerea polinoamelor, un câmp pentru afișarea rezultatului și etichete pentru a indica fiecare câmp. De asemenea, creăm un panou separat pentru butoanele de operații, care conține butoanele dispuse într-o grilă de 4x2. De asemenea, un alt panou separat este creat pentru butonul de ștergere, care este adăugat în partea de jos a panoului de butoane.

În ceea ce privește implementarea logicii de calcul, metoda actionPerformed() se ocupă de preluarea datelor de intrare și de calculul rezultatului pentru fiecare operație. Fiecare buton de operație este asociat cu un bloc de instrucțiuni if-else care verifică care buton a fost apăsat și calculează rezultatul corespunzător folosind clasa Operations din pachetul org.example.BussinesLogic.

La finalul procesului de programare, GUI-ul este prezentat utilizatorului prin intermediul metodei setVisible(true). Acesta așteaptă până când utilizatorul introduce datele de intrare și apasă butoanele necesare pentru a obține rezultatele dorite. În cadrul aceleiași clase, este definită și o metodă statică "main", care crează o nouă instanță a clasei GUI



5. Rezultate

Am realizat teste unitare folosind JUnit pentru a verifica corectitudinea funcționării calculatorului de polinoame. Aceste teste au acoperit funcționalitățile de bază ale calculatorului, cum ar fi adunarea, scăderea, înmulțirea, împărțirea, derivarea și integrarea polinoamelor. Pentru fiecare operație în parte, am furnizat atât un exemplu corect, cât și unul incorect, pentru a verifica modul în care programul gestionează situațiile greșite.

Testarea adunării

```
@Test
void testAdd() {
    Polynomial p1 = new Polynomial("-x^3+2x^2-5x+4");
    Polynomial p2 = new Polynomial("-4x^2-5");
    Polynomial expected = new Polynomial("-x^3-2.0x^2-5.0x-1.0");
    assertEquals(expected.toString(), Operations.add(p1, p2).toString());
}

@Test
void testAddg() {
    Polynomial p1 = new Polynomial("-x^3+2x^2-5x+4");
    Polynomial p2 = new Polynomial("-4x^2-5");
    Polynomial wrong = new Polynomial("-x^3-2.0x^2-5.0x+1.0");
    assertEquals(wrong.toString(), Operations.add(p1, p2).toString());
}
```

Testarea scăderii

```
@Test
void testSub() {
    Polynomial p1 = new Polynomial("-x^3+2x^2-5x+4");
    Polynomial p2 = new Polynomial("-4x^2-5");
    Polynomial expected = new Polynomial("-x^3+6.0x^2-5.0x+9.0");
    assertEquals(expected.toString(), Operations.sub(p1, p2).toString());
}

@Test
void testSubg() {
    Polynomial p1 = new Polynomial("-x^3+2x^2-5x+4");
    Polynomial p2 = new Polynomial("-4x^2-5");
    Polynomial wrong = new Polynomial("x^3+6.0x^2-5.0x");
    assertEquals(wrong.toString(), Operations.sub(p1, p2).toString());
}
```

Testarea înmulțirii

```java
@Test
void testMul() {
    Polynomial p1 = new Polynomial("-x^3+2x^2-5x+4");
    Polynomial p2 = new Polynomial("-4x^2-5");
    Polynomial expected = new Polynomial("4.0x^5-8.0x^4+25.0x^3-26.0x^2+25.0x-20.0");
    assertEquals(expected.toString(), Operations.mul(p1, p2).toString());
}

@Test
void testMulg() {
    Polynomial p1 = new Polynomial("-x^3+2x^2-5x+4");
    Polynomial p2 = new Polynomial("-4x^2-5");
    Polynomial wrong = new Polynomial("4.0x^5-8.0x^4+25.0x^3+25.0x+20.0");
    assertEquals(wrong.toString(), Operations.mul(p1, p2).toString());
}
```

Testarea împărțirii

```java
@Test
public void testDiv() {
    Polynomial p1 = new Polynomial("x^3-2x^2+6x-5");
    Polynomial p2 = new Polynomial("x^2-1");

    Polynomial[] result = Operations.div(p1, p2);
    Polynomial quotient = result[0];
    Polynomial remainder = result[1];

    Polynomial expectedQuotient = new Polynomial("x-2");
    Polynomial expectedRemainder = new Polynomial("7.0x-7.0");

    assertEquals(expectedQuotient.toString(), quotient.toString());
    assertEquals(expectedRemainder.toString(), remainder.toString());
}

@Test
public void testDivg() {
    Polynomial p1 = new Polynomial("x^3-2x^2+6x-5");
    Polynomial p2 = new Polynomial("x^2-1");

    Polynomial[] result = Operations.div(p1, p2);
    Polynomial quotient = result[0];
    Polynomial remainder = result[1];

    Polynomial wrongQuotient = new Polynomial("x-2");
    Polynomial wrongRemainder = new Polynomial("7.0x+7.0");

    assertEquals(wrongQuotient.toString(), quotient.toString());
    assertEquals(wrongRemainder.toString(), remainder.toString());
}
```

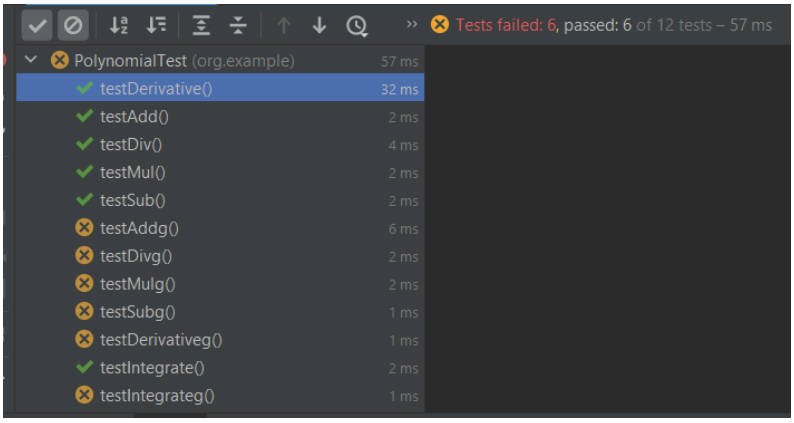
Testarea derivării

```java
@Test
void testDerivative() {
    Polynomial p = new Polynomial("2x^3+3x^2+5x+7");
    Polynomial expected = new Polynomial("6.0x^2+6.0x+5.0");
    assertEquals(expected.toString(), Operations.derivative(p).toString());
}

@Test
void testDerivativeg() {
    Polynomial p = new Polynomial("2x^3+3x^2+5x+7");
    Polynomial wrong = new Polynomial("6.0x^3+6.0x+5.0");
    assertEquals(wrong.toString(), Operations.derivative(p).toString());
}
```

Testarea integrării

```java
@Test
void testIntegrate() {
    Polynomial p = new Polynomial("2x^3+3x^2+5x+7");
    Polynomial expected = new Polynomial("0.5x^4+x^3+2.5x^2+7.0x");
    assertEquals(expected.toString(), Operations.integrate(p).toString());
}

@Test
void testIntegrateg() {
    Polynomial p = new Polynomial("2x^3+3x^2+5x+7");
    Polynomial wrong = new Polynomial("0.55x^4+1.0x^3+2.5x^2+7.0x");
    assertEquals(wrong.toString(), Operations.integrate(p).toString());
}
```

Rezultate



În urma testelor efectuate, am constatat ca calculatorul de polinoame implementat funcționează corect. Prin urmare, testarea unitară cu ajutorul utilitarului JUnit a fost esențială pentru dezvoltarea unui calculator de polinoame fiabil. Aceasta a asigurat ca aplicația funcționează corect si că poate fi utilizată in mod eficient si intuitiv de către utilizatori.

6. Concluzii

În concluzie, realizarea acestui calculator de polinoame a reprezentat o provocare tehnică interesantă, dar în același timp și oportunitatea de a învăța mai multe despre matematica polinoamelor și programarea orientată pe obiecte. Am creat un sistem care poate efectua o gamă largă de operații pe polinoame și am dezvoltat o interfață grafică intuitivă și eficientă pentru utilizator.

În ceea ce privește dezvoltarea ulterioara a acestui proiect, se poate lua în considerare implementarea de noi funcționalități precum găsirea rădăcinilor, simplificarea polinoamelor sau introducerea de polinoame cu mai multe variabile. De asemenea, se poate îmbunătăți interfața grafică pentru a fi și mai prietenoasă și accesibilă utilizatorului.

În general, acest proiect mi-a permis să îmi dezvolt abilitătile tehnice si cunoștințele de matematică, precum și să obțin o experiență practică în proiectarea și dezvoltarea de sisteme de calcul.

7. Bibliografie

1. Java Regular Expressions (https://www.w3schools.com/java/java_regex.asp)
2. Decode polynomial from String with Pattern and Matcher (https://stackoverflow.com/questions/34946528/decode-polynomial-from-string-with-pattern-and-matcher)
3. Java for Scientific Computing: Polynomial Division (https://www.youtube.com/watch?v=i_Qn-d6WExs)
4. Operations on Polynomials (https://courses.lumenlearning.com/beginalgebra/chapter/4-2-2-adding-and-subtracting-polynomials/)
5. Regexr