CALCULATOR DE POLINOAME

DOCUMENTATIE - TEMA 1 - TP

PĂCURAR IRINA
UTCN, CTI, SERIA A, GRUPA 30223
ÎNDRUMĂTORI: PROF. DR. ING. TUDOR CIOARĂ, OVIDIU MAJA
- martie 2024 -

Cuprins

1. Obiectivul temei

Scopul principal al proiectului este să dezvolte o aplicație de calculator pentru polinoame, oferind utilizatorilor posibilitatea de a introduce polinoame și de a efectua diverse operații matematice specifice acestora. Această aplicație va fi utilă în contextul educațional sau pentru persoanele care lucrează cu polinoame în diverse domenii științifice sau tehnice.

Obiectivele secundare ale proiectului sunt multiple. În primul rând, se dorește familiarizarea cu structurile de date din Java, cu accent pe lucrul cu map-uri, care sunt utile pentru gestionarea polinoamelor și a datelor asociate acestora. De asemenea, se urmărește scrierea unui cod clar și ușor de înțeles, promovând astfel bunele practici de programare și îmbunătățind calitatea soluției dezvoltate. Organizarea codului în pachete este esențială pentru menținerea structurii proiectului și pentru a facilita colaborarea între membrii echipei sau pentru a permite extinderea și modificarea ulterioară a aplicației.

Un alt obiectiv secundar important este testarea exhaustivă a funcționalităților aplicației folosind JUnit. Testarea este esențială pentru a asigura funcționarea corectă a tuturor componentelor și pentru a identifica și rezolva eventualele erori sau probleme înainte de a lansa aplicația în producție.

Procesul de dezvoltare va implica mai multe etape, începând cu crearea modelelor necesare pentru reprezentarea polinoamelor și a operațiilor pe acestea. Urmează proiectarea și implementarea funcționalităților specifice polinoamelor, precum adunarea, scăderea, înmulțirea, împărțirea și evaluarea acestora. Dezvoltarea unei interfețe grafice intuitive și ușor de utilizat va fi, de asemenea, o parte importantă a proiectului, pentru a oferi o experiență plăcută utilizatorilor.

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

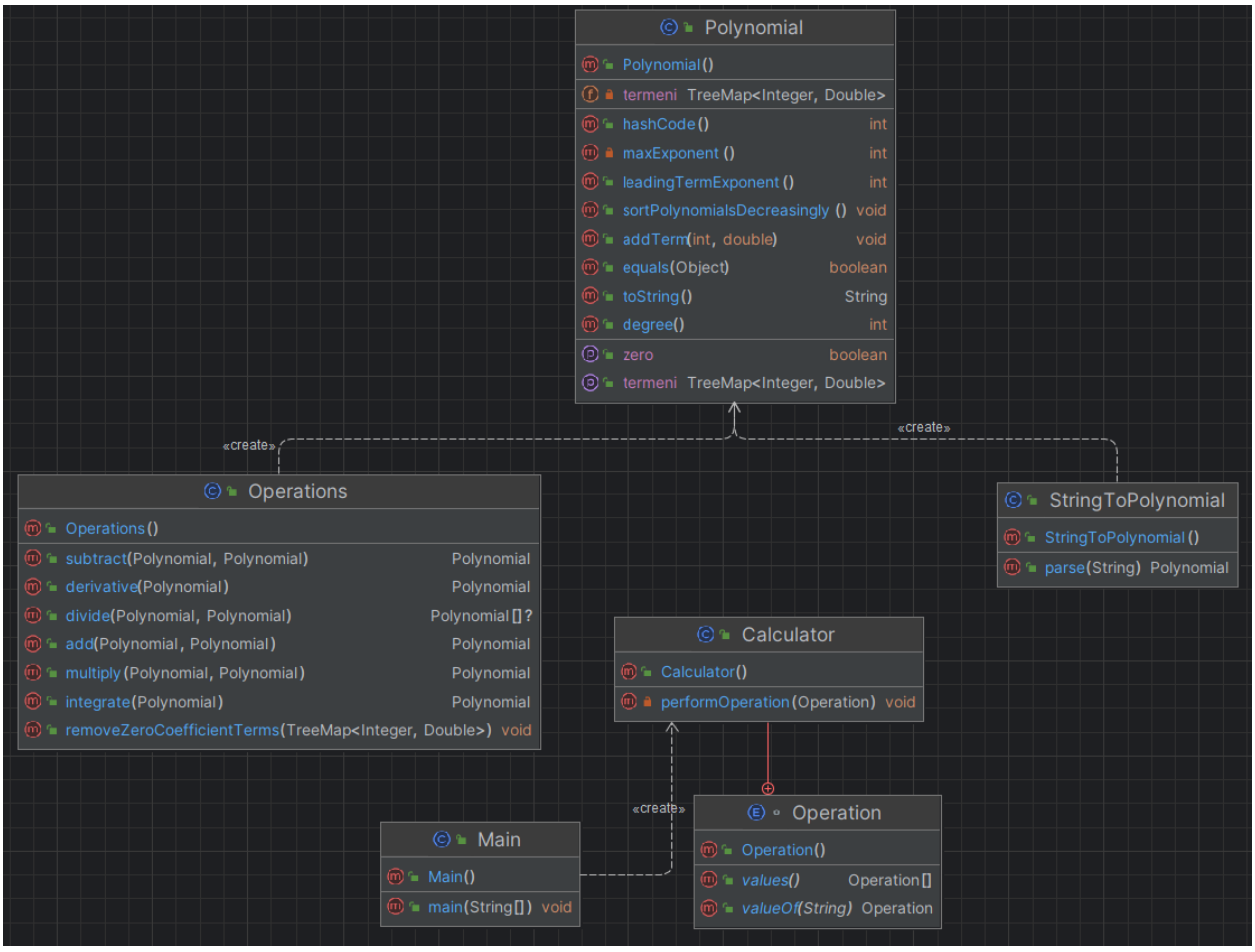
În cadrul analizei problemei și modelării, trebuie să identificăm cerințele funcționale și non-funcționale ale aplicației noastre de calculator pentru polinoame. Cerințele funcționale includ capacitatea de a citi polinoame, de preferință utilizând expresii regulate pentru a asigura o introducere corectă a datelor, și de a efectua o serie de operații matematice specifice asupra acestora, inclusiv adunare, scădere, înmulțire, împărțire, derivare și integrare. Cerințele non-funcționale vizează eficiența operațiilor menționate și crearea unei interfețe intuitive și ușor de folosit.

În ceea ce privește cazurile de utilizare, principalul scenariu implică introducerea a unuia sau a două polinoame în câmpurile designate, selectarea operației dorite și executarea acesteia prin apăsarea butonului corespunzător, urmată de afișarea rezultatului. Utilizatorul trebuie să aibă posibilitatea de a reintroduce rezultatul într-unul dintre câmpurile de date sau de a-l copia în clipboard pentru utilizare ulterioară.

Dezvoltarea unei astfel de aplicații va necesita implementarea unor modele de date corespunzătoare pentru polinoame, precum și logica necesară pentru operațiile matematice. Interfața grafică trebuie proiectată astfel încât să faciliteze introducerea datelor și selectarea operațiilor, iar testarea riguroasă a tuturor funcționalităților este esențială pentru asigurarea corectitudinii și fiabilității aplicației.

3. Proiectare

Aplicația include 5 clase principale: *MainClass*, aflată în pachetul *main*, responsabilă de inițializarea interfeței grafice; clasa *Calculator*, din cadrul pachetului *GUI*, care conține toate elementele grafice-vizuale și facilitează interacțiunea utilizatorului cu aplicația; *Operations*, o clasă cu metode statice ce conține toate operațiile apelate de butoanele din GUI, *StringToPolynomial* ce include parsarea input-ului, ambele făcând parte din pachetul *logic* și clasa *Polynomial*, modelul principal al aplicației, în care sunt memorate obiectele de tip polinom.



4. *Implementare*

4.1. Clasa Main

Clasa main în acest context este responsabilă pentru inițializarea și afișarea aplicației calculatorului de polinoame. În codul dat, se folosește metoda invokeLater() din clasa SwingUtilities, care este utilizată pentru a programa o acțiune (în acest caz, crearea și afișarea calculatorului) pentru a fi executată pe firul de dispecerat al interfeței grafice Swing.

Această abordare este importantă în aplicațiile Swing pentru a asigura că toate operațiunile legate de interfața grafică sunt efectuate pe firul de dispecerat al interfeței grafice Swing, ceea ce ajută la evitarea blocării interfeței grafice și asigură o experiență fluidă pentru utilizator. Astfel, obiectul calculatorului este creat și afișat în cadrul metodei run(), care este apelată în mod asincron pe firul de dispecerat al interfeței grafice.

```
public class Main {
    Irina25P *
    public static void main(String[] args) {

        Irina25P
        SwingUtilities.invokeLater(new Runnable() {
            Irina25P
            @Override
            public void run() {
                Calculator calculator = new Calculator();
                calculator.setVisible(true);
            }
        });
    }
}
```

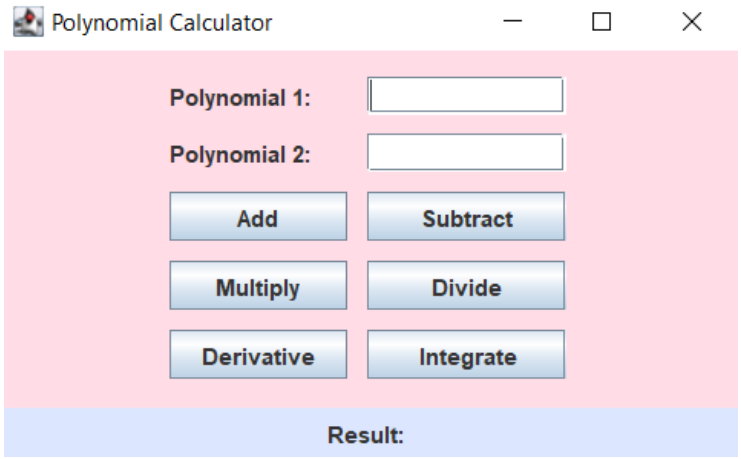
4.2. GUI

Interfața reprezintă un calculator pentru operații cu polinoame. Aceasta conține:

1. Câmpuri de introducere a polinoamelor:
   - Există două câmpuri de text unde utilizatorul poate introduce coeficienții și exponenții polinoamelor.
   - Primul câmp este pentru Polynomial 1, iar al doilea pentru Polynomial 2.
2. Butoane pentru operații:
   - Există șase butoane care permit utilizatorului să selecteze operația dorită: Adunare, Scădere, Înmulțire, Împărțire, Derivare și Integrare.
   - Fiecare buton are un text corespunzător operației pe care o reprezintă.
3. Panou de afișare a rezultatului:
   - Sub câmpurile de introducere a polinoamelor, se află un panou de afișare a rezultatului.

- Aici va fi afișat rezultatul operației selectate sau un mesaj de eroare în cazul unei operații invalide sau a unei împărțiri la zero.

4. Operații suportate:
   - Adunare: Adună două polinoame.
   - Scădere: Scade un polinom din altul.
   - Înmulțire: Înmulțește două polinoame.
   - Împărțire: Împarte un polinom la altul.
   - Derivare: Calculează derivata unui polinom.
   - Integrare: Calculează integrala unui polinom.



4.3. Polynomial

Clasa Polynomial este responsabilă pentru reprezentarea și manipularea polinoamelor în cadrul programului. Iată o scurtă descriere a clasei:

1. Variabilă de instanță:
   - *termeni*: Este un obiect de tip TreeMap<Integer, Double> care stochează coeficienții polinomului împreună cu exponenții lor. Cheile TreeMap-ului reprezintă exponenții, iar valorile reprezintă coeficienții.
2. Constructori:
   - *Polynomial()*: Constructorul implicit care inițializează un polinom gol.
   - *Polynomial(Map<Integer, Double> termeni)*: Constructor care primește un Map de termeni și inițializează polinomul cu acești termeni.
3. Metode:
   - *getTermeni()*: Returnează termenii polinomului sub formă de TreeMap<Integer, Double>.
   - *setTermeni(TreeMap<Integer, Double> termeni)*: Setează termenii polinomului cu cei din termeni.
   - *degree()*: Calculează gradul polinomului, adică cel mai mare exponent din termenii săi.
   - *leadingTermExponent()*: Returnează exponentul termenului principal al polinomului.
   - *isZero()*: Verifică dacă polinomul este zero (adică nu are niciun termen).

- *addTerm(int exponent, double coefficient)*: Adaugă un termen nou la polinom cu un anumit exponent și coeficient, înlocuind termenul existent dacă acesta deja există.
- *sortPolynomialsDecreasingly()*: Sortează termenii polinomului în ordine descrescătoare a exponentului.
- *maxExponent()*: Returnează cel mai mare exponent din termenii polinomului.
- *toString()*: Returnează o reprezentare sub formă de șir de caractere a polinomului.

Această clasă permite lucrul cu polinoame prin adăugarea și sortarea termenilor, calculul gradului, a termenului principal, a verificării dacă este zero, precum și conversia la șir de caractere pentru afișare sau manipulare ulterioară.

```
public class Polynomial {
    15 usages
    private TreeMap<Integer, Double> termeni;
    Irina25P
    @Override
    public boolean equals(Object o) {...}
    Irina25P
    @Override
    public int hashCode() { return Objects.hash(termeni); }
    9 usages  Irina25P
    public Polynomial() { termeni = new TreeMap<>(); }
    24 usages  Irina25P
    public TreeMap<Integer, Double> getTermeni() { return termeni; }
    1 usage  Irina25P
    public int degree() {...}
    3 usages  Irina25P
    public int leadingTermExponent() {...}
    1 usage  Irina25P
    public boolean isZero() { return termeni.isEmpty(); }
    1 usage  Irina25P
    public void addTerm(int exponent, double coefficient) {...}
    8 usages  Irina25P
    public void sortPolynomialsDecreasingly() {...}
    1 usage  Irina25P
    private int maxExponent() {...}
    Irina25P
    @Override
    public String toString() {...}
}
```

4.4. Operations

Clasa Operations conține metode pentru efectuarea diferitelor operații matematice pe polinoame. Iată o descriere pentru fiecare metodă:

1. *add(Polynomial polynomial1, Polynomial polynomial2):*
   - Această metodă efectuează adunarea a două polinoame, polynomial1 și polynomial2.
   - Se creează un polinom nou (result) pentru a stoca rezultatul.
   - Se adaugă termenii polinomului 1 în result.
   - Se parcurg termenii polinomului 2 și se adaugă coeficienții la termenii corespunzători în result.
   - Se elimină termenii cu coeficientul zero din result și se sortează descrescător după exponent.

2. *removeZeroCoefficientTerms(TreeMap<Integer, Double> polynomials):*
   - Această metodă elimină termenii cu coeficientul zero dintr-un polinom dat, reprezentat sub formă de TreeMap cu chei de tip integer și valori de tip double.

3. *subtract(Polynomial polynomial1, Polynomial polynomial2):*
   - Această metodă efectuează scăderea a două polinoame, polynomial1 și polynomial2.
   - Se creează un polinom nou (result) pentru a stoca rezultatul.
   - Se adaugă termenii polinomului 1 în result.
   - Se parcurg termenii polinomului 2 și se scad coeficienții de la termenii corespunzători din result.
   - Se elimină termenii cu coeficientul zero din result și se sortează descrescător după exponent.

4. *multiply(Polynomial polynomial1, Polynomial polynomial2):*
   - Această metodă efectuează înmulțirea a două polinoame, polynomial1 și polynomial2.
   - Se creează un polinom nou (result) pentru a stoca rezultatul.
   - Se parcurg toți termenii polinomului 1 și polinomului 2 și se înmulțesc coeficienții și se adună exponenții pentru a forma termenii noului polinom result.
   - Se sortează descrescător după exponent termenii polinomului rezultat.

5. *derivative(Polynomial polynomial1):*
   - Această metodă calculează derivata unui polinom dat polynomial1.
   - Se creează un polinom nou (result) pentru a stoca rezultatul.
   - Pentru fiecare termen din polynomial1, se calculează noua putere și noul coeficient după regula derivării, și se adaugă la result.
   - Se sortează descrescător după exponent termenii polinomului rezultat.

6. *integrate(Polynomial polynomial1):*
   - Această metodă calculează integrala unui polinom dat polynomial1.
   - Se creează un polinom nou (result) pentru a stoca rezultatul.
   - Pentru fiecare termen din polynomial1, se calculează noua putere și noul coeficient după regula integrării, și se adaugă la result.
   - Se sortează descrescător după exponent termenii polinomului rezultat.

7. *divide(Polynomial polynomial1, Polynomial polynomial2)*:
    - Această metodă efectuează împărțirea a două polinoame, polynomial1 și polynomial2.
    - Se sortează descrescător după exponent termenii ambelor polinoame.
    - Se parcurge polinomul 1 și se împarte termenii săi la termenii polinomului 2.
    - Se actualizează polinomul 1 cu restul și se continuă procesul până când gradul polinomului 1 devine mai mic decât gradul polinomului 2.
    - Se returnează un vector de polinoame, unde primul element este rezultatul împărțirii (câtul) și al doilea element este restul.

```
public class Operations {
    Irina25P
    public static Polynomial add(Polynomial polynomial1, Polynomial polynomial2) {...}

    2 usages  Irina25P
    public static void removeZeroCoefficientTerms(TreeMap<Integer, Double> polynomials) {...}

    4 usages  Irina25P
    public static Polynomial subtract(Polynomial polynomial1, Polynomial polynomial2) {...}

    4 usages  Irina25P
    public static Polynomial multiply(Polynomial polynomial1, Polynomial polynomial2) {...}

    3 usages  Irina25P
    public static Polynomial derivative(Polynomial polynomial1) {...}

    3 usages  Irina25P
    public static Polynomial integrate(Polynomial polynomial1) {...}

    3 usages  Irina25P
    public static Polynomial[] divide(Polynomial polynomial1, Polynomial polynomial2) {...}
}
```

4.5. StringToPolynomial

Funcția parse(String polynomial) este responsabilă pentru parsarea unui șir de caractere care reprezintă un polinom într-un obiect de tip Polynomial.

Se definește un șablon de expresie regulată folosind clasa Pattern. Acest șablon este conceput pentru a potrivi termenii unui polinom, inclusiv coeficienți și exponenți. Se crează un obiect Matcher pentru a căuta termenii polinomului în șirul de caractere polynomial. Se parcurg termenii polinomului folosind metoda find() a obiectului Matcher. Se extrage coeficientul și exponentul fiecărui termen. Pentru coeficient, se verifică mai întâi dacă este null sau gol. Dacă da, se verifică dacă este primul termen al polinomului. Dacă da, coeficientul este 1.0, altfel este 0.0. Dacă coeficientul este "+" sau "-", acesta este interpretat ca 1.0, respectiv -1.0. În caz contrar, este

convertit la dublă folosind Double.parseDouble(). Pentru exponent, se verifică dacă este null. Dacă da, se verifică dacă există "x" în termen. Dacă da, exponentul este 1, altfel este 0. Dacă exponentul nu este null, este convertit la int folosind Integer.parseInt(). Dacă exponentul nu este 0 sau coeficientul nu este 0, termenul este adăugat la polinomul rezultat folosind metoda addTerm() a obiectului Polynomial. La final, se returnează polinomul rezultat.

Această funcție parsează șirurile de caractere care reprezintă polinoame în forma standard, cum ar fi "3x^2-2x+5". Este capabilă să gestioneze coeficienți întregi sau zecimali, precum și exponenți întregi pozitivi.

```
public static Polynomial parse(String polynomial) {
    Polynomial result = new Polynomial();
    Pattern pattern = Pattern.compile( regex: "((?:[+-]?\\d*\\.?\\d*)?)(x(\\^(\\d+))?)?");
    Matcher matcher = pattern.matcher(polynomial);

    boolean firstTerm = true;

    while (matcher.find()) {
        String coefficientInit = matcher.group(1);
        String exponentInit = matcher.group(4);

        double coeff;
        if (coefficientInit == null || coefficientInit.isEmpty()) {
            if (firstTerm) {...} else {
                coeff = 0.0;
            }
        } else if (coefficientInit.equals("+")) {...} else if (coefficientInit.equals("-")) {...} else {
            coeff = Double.parseDouble(coefficientInit);
        }

        int exp;
        if (exponentInit == null) {...} else {
            exp = Integer.parseInt(exponentInit);
        }
        if (exp != 0 || coeff != 0) {...}
        if (firstTerm) {...}
    }

    return result;
}
```

5. *Rezultate*

Aplicația a fost supusă unor teste riguroase pe diverse tipuri de intrări în timpul dezvoltării. Testele au inclus scenarii precum: Polinoame constant, polinoame care încep cu coeficienți negativi, polinoame cu termeni nuli sau cu coeficienți 0, polinoame cu termeni negative, polinoame care nu conțin constante intrări goale sau invalide. Aceste teste au fost concepute pentru a asigura că aplicația gestionează corect diversele scenarii și oferă rezultate valide în toate cazurile.

De asemenea, s-au efectuat teste unitare folosind framework-ul JUnit. În aceste teste, au fost inițializate două polinoame cu valori cunoscute, iar rezultatele operațiilor pe aceste polinoame au fost comparate cu rezultatele așteptate. Aceasta a permis validarea metodelor și asigurarea funcționării corecte a calculatorului, confirmând că operațiile de adunare, scădere, înmulțire, derivare, integrare și împărțire sunt implementate corespunzător.

```
public class Tests {
    2 usages
    private static Integer numberOfTests = 0;
    13 usages
    private static Integer numberOfTestsPassed = 0;
    2 usages
    private static Operations operations;

    Irina25P
    @BeforeClass
    public static void init() {...}

    Irina25P
    @AfterClass
    public static void finalMethod() {...}
    Irina25P
    @Before
    public void increment() { numberOfTests++; }
    Irina25P
    @After
    public void afterEachTest() { System.out.println("Test Finished"); }
    Irina25P
    @Test
    public void testAddition() {
        Polynomial p1 = StringToPolynomial.parse( polynomial: "2x^2+3x+1");
        Polynomial p2 = StringToPolynomial.parse( polynomial: "4x^3+2x^2-x");
        Polynomial expected = StringToPolynomial.parse( polynomial: "4x^3+4x^2+2x+1");
        assertEquals(expected, Operations.add(p1, p2));
        numberOfTestsPassed++;
    }
```

6. *Concluzii și dezvoltări ulterioare*

Proiectul poate fi extins și îmbunătățit în mai multe moduri: adăugarea unei opțiuni pentru a permite utilizatorului să aleagă între teme de culoare întunecată sau luminoasă poate îmbunătăți experiența de utilizare și personalizarea aplicației. De asemenea, îmbunătățirea mecanismului de highlight pentru inputul greșit poate facilita identificarea erorilor și oferi feedback mai clar și mai detaliat utilizatorului. Actualizarea aspectului GUI cu un design mai modern și atrăgător poate îmbunătăți atractivitatea și utilizabilitatea aplicației. Implementarea altor operații matematice sau funcționale poate extinde capabilitățile calculatorului și îl poate face mai util și mai versatil pentru utilizatori. Dezvoltarea unei funcționalități care să permită utilizatorului să introducă direct operațiile într-un singur câmp de input, care să fie parsat și rezolvat automat în timp real, poate oferi o experiență mai fluidă și mai rapidă a calculatorului. Acest proiect a avut un rol important în familiarizarea cu conceptele de programare orientată pe obiecte (OOP), organizarea codului în pachete și principiile de dezvoltare a aplicațiilor software. De asemenea, a contribuit la îmbunătățirea abilităților practice în lucrul logic și ordonat în cadrul dezvoltării unei aplicații, respectând principiile OOP și asigurând o structură bine definită și modulară a codului.

7. *Bibliografie*

- Fundamental Programming Techniques (dsrl.eu)
- Assertions (JUnit 5.10.2 API)
- Google Java Style Guide
- Trail: Creating a GUI With Swing (The Java™ Tutorials) (oracle.com)
- A Guide to JUnit 5 | Baeldung
- java - Regex for polynomial expression - Stack Overflow
- Java Swing Tutorial - javatpoint
- regex101: build, test, and debug regex