DOCUMENTATIE

TEMA 1

NUME STUDENT: MATIȘ OANA-ANTONIA
GRUPA: 30228

CUPRINS

1. Obiectivul temei

Obiectivul principal al acestei teme este proiectarea și implementarea unui calculator de polinoame, care dispune de o interfață grafică adecvat adaptată. Utilizatorul va putea să introducă două polinoame de la tastatură, să selecteze operația matematică dorită (adunare/scădere/înmulțire/împărțire/derivare/integrare) și să vizualizeze rezultatul pe ecranul special conceput.

Obiectivele secundare ale acestei teme sunt:

- **Analiza problemei** - Capitolul 2

Această etapă implică examinarea problemei polinoamelor, inclusiv definirea conceptelor-cheie si a termenilor utilizati in contextul problemei. Aici, se va descrie, de asemenea, necesitatea si utilitatea unui calculator de polinoame si se va discuta despre metodele de reprezentare a polinoamelor si operatiile aritmetice pe care calculatorul ar trebui sa le efectueze.

- **Proiectarea calculatorului de polinoame** - Capitolul 3

Această etapă implică definirea specificațiilor și cerintelor calculatorului de polinoame, precum si identificarea metodelor si algoritmilor adecvati pentru realizarea acestuia. În acest capitol se vor descrie modul de organizare a datelor, modul de implementare a operatiilor si a functiilor, precum si modul de interactiune cu utilizatorul prin intermediul interfetei grafice.

- **Implementarea calculatorului de polinoame** - Capitolul 4

Această etapă implică implementarea codului sursă al calculatorului de polinoame, conform specificatiilor si cerintelor definite in capitolul 3. In aceasta sectiune se vor detalia aspectele tehnice ale implementarii si se vor prezenta fragmente de cod relevante.

- **Testarea calculatorului de polinoame** - Capitolul 5

Această etapă implică testarea si validarea calculatorului de polinoame pentru a verifica daca functioneaza conform specificatiilor si cerintelor definite in capitolul 3. In acest capitol se vor descrie scenariile de testare si se vor prezenta rezultatele obtinute.

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

2.1. Cerințe funcționale

- Calculatorul de polinoame ar trebui să permită utilizatorilor să introducă două polinoame.
- Calculatorul de polinoame ar trebui să permită utilizatorilor să selecteze operația matematică.
- Calculatorul de polinoame ar trebui să permită adunarea a două polinoame.
- Calculatorul de polinoame ar trebui să permită scăderea a două polinoame.
- Calculatorul de polinoame ar trebui să permită înmulțirea a două polinoame.
- Calculatorul de polinoame ar trebui să permită împărțirea a două polinoame.
- Calculatorul de polinoame ar trebui să permită derivarea unui polinom.
- Calculatorul de polinoame ar trebui să permită integrarea unui polinom.

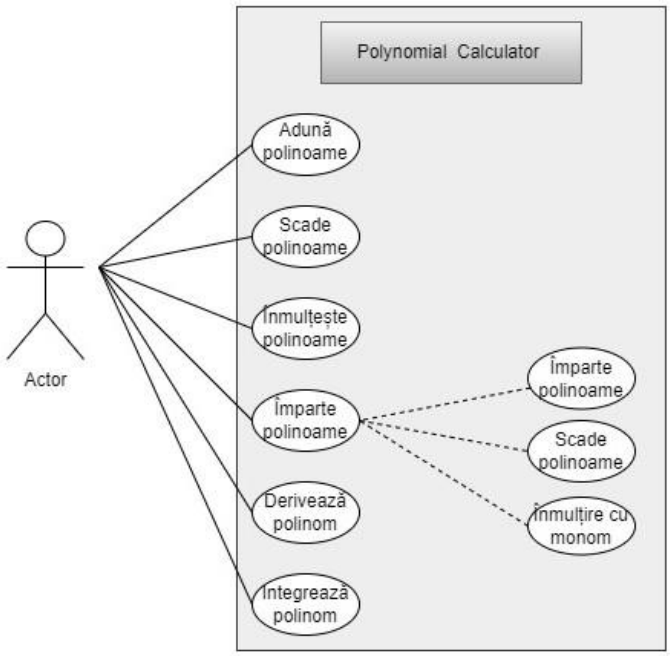
2.2. Cerințe non funcționale

- Calculatorul de polinoame ar trebui să fie intuitiv și ușor de folosit de către utilizatori
- Calculatorul de polinoame ar trebui să aibă o interfață interactivă și receptive
- Calculatorul de polinoame ar trebui să afișeze mesaje de eroare în cazul în care utilizatorul nu introduce corespunzător polinoamele.
- Calculatorul de polinoame ar trebui să fie precis și să efectueze corect operațiile matematice.

2.3. Use cases

Voi prezenta diagramele use-case pentru cazurile de utilizare ale unei aplicații de efectuare a operațiilor matematice asupra polinoamelor, unde actorul este reprezentat de orice utilizator al aplicației care dorește să efectueze operații matematice asupra polinoamelor.

Diagrama use-case este o modalitate grafică de a descrie interacțiunea între actori și sistemul de software, identificându-se acțiunile specifice pe care utilizatorii le pot efectua pentru a realiza sarcinile lor în aplicație.



1) Use case: adunarea polinoamelor
Primary Actor: user (utilizator)
Main Succes Scenario:
1) Utilizatorul inserează două polinoame în câmpurile adecvate ale interfeței grafice.
2) Utitilizatorul selectează operația “Add” din meniul de butoane.
3) Calculatorul de polinoame efectuează adunarea celor două polinoame și afișează rezultatul in text field-ul adecvat.

Alternative sequence:
- Utilizatorul inserează două polinoame incorecte.
- Scenariul revine la pasul 1.

2) Use case: scăderea polinoamelor
Primary Actor: user (utilizator)
Main Succes Scenario:
1) Utilizatorul inserează două polinoame în câmpurile adecvate ale interfeței grafice.
2) Utitilizatorul selectează operația "Substract" din meniul de butoane.
3) Calculatorul de polinoame efectuează scăderea celor două polinoame și afișează rezultatul in text field-ul adecvat.

Alternative sequence:
- Utilizatorul inserează două polinoame incorecte.
- Scenariul revine la pasul 1.

3) Use case: înmulțirea polinoamelor
Primary Actor: user (utilizator)
Main Succes Scenario:
1) Utilizatorul inserează două polinoame în câmpurile adecvate ale interfeței grafice.
2) Utitilizatorul selectează operația "Multiply" din meniul de butoane.
3) Calculatorul de polinoame efectuează înmulțirea celor două polinoame și afișează rezultatul in text field-ul adecvat.

Alternative sequence:
- Utilizatorul inserează două polinoame incorecte.
- Scenariul revine la pasul 1.

4) Use case: împărțirea polinoamelor
Primary Actor: user (utilizator)
Main Succes Scenario:
1) Utilizatorul inserează două polinoame în câmpurile adecvate ale interfeței grafice.
2) Utilizatorul selectează operația "Divide" din meniul de butoane.
3) Calculatorul de polinoame efectuează adunarea celor două polinoame și afișează rezultatul in text field-ul adecvat.

Alternative sequence:
- Utilizatorul inserează două polinoame incorecte.
- Scenariul revine la pasul 1.

5) Use case: derivarea polinoamelor
Primary Actor: user (utilizator)
Main Succes Scenario:
1) Utilizatorul inserează un polinom în primul câmp al interfeței grafice.
2) Utitilizatorul selectează operația "Derive" din meniul de butoane.
3) Calculatorul de polinoame efectuează derivarea polinomului și afișează rezultatul in text field-ul adecvat.

Alternative sequence:

- Utilizatorul inserează un polinom incorect.
- Scenariul revine la pasul 1.

6) Use case: adunarea polinoamelor
Primary Actor: user (utilizator)
Main Succes Scenario:
1) Utilizatorul inserează un polinom în primul câmp al interfeței grafice.
2) Ututilizatorul selectează operația “Integrate” din meniul de butoane.
3) Calculatorul de polinoame efectuează integrarea polinomului și afișează rezultatul in text field-ul adecvat.

Alternative sequence:
- Utilizatorul inserează un polinom incorect.
- Scenariul revine la pasul 1.

3. Proiectare

Acest capitol va prezenta proiectarea aplicației din perspectiva paradigmei de programare orientate pe obiecte și pașii pe care i-am făcut în acest sens. De asemenea, voi expune diagramele UML de clase și pachete, precum și structurile de date și algoritmi utilizați în implementarea aplicației.

3.1) Proiectarea OOP

Proiectarea OOP a unei aplicații Java Swing implică definirea de clase, obiecte, metode și interfețe pentru a crea o interfață grafică de utilizator (GUI) interactivă. Principiile OOP precum încapsularea, moștenirea și polimorfismul sunt utilizate în mod obișnuit pentru a crea un design modular și scalabil.

Pentru a crea interfața grafică a aplicației, sunt utilizate clasele și interfețele din pachetul Swing, precum JFrame, JPanel, JButton, JTextArea și altele. Acestea sunt apoi extinse și personalizate pentru a se potrivi nevoilor specifice ale aplicației.

În ceea ce privește diagramele UML, acestea pot fi utilizate pentru a ilustra clasele, interfețele și relațiile dintre acestea în cadrul aplicației. De asemenea, diagramele UML de pachete pot fi utilizate pentru a grupa și organiza clasele și interfețele în funcție de funcționalitatea lor.

Aplicatia Calculator de Polinoame contine 4 clase:

- Main

În interiorul metodei main() este creată o instanță a clasei Interface, care este clasa principală pentru interfața grafică a aplicației Polynomial Calculator. De asemenea, se inițializează și afișează fereastra principală a aplicației prin apelarea metodei setVisible() a instanței window. Clasa Main este esențială pentru pornirea corectă a aplicației și pentru a permite utilizatorului să interacționeze cu interfața grafică.

- Polynomial

Clasa Polynomial reprezintă o implementare a unui polinom, utilizând un Map de exponenți și coeficienți pentru a stoca termenii polinomului. Aceasta conține metode pentru

adăugarea de termeni, obținerea coeficientului unui anumit exponent, calcularea gradului polinomului, eliminarea termenilor cu coeficientul zero și transformarea polinomului într-o reprezentare sub formă de string. Clasa mai include, de asemenea, o metodă pentru a verifica dacă polinomul este nul.

- Operații

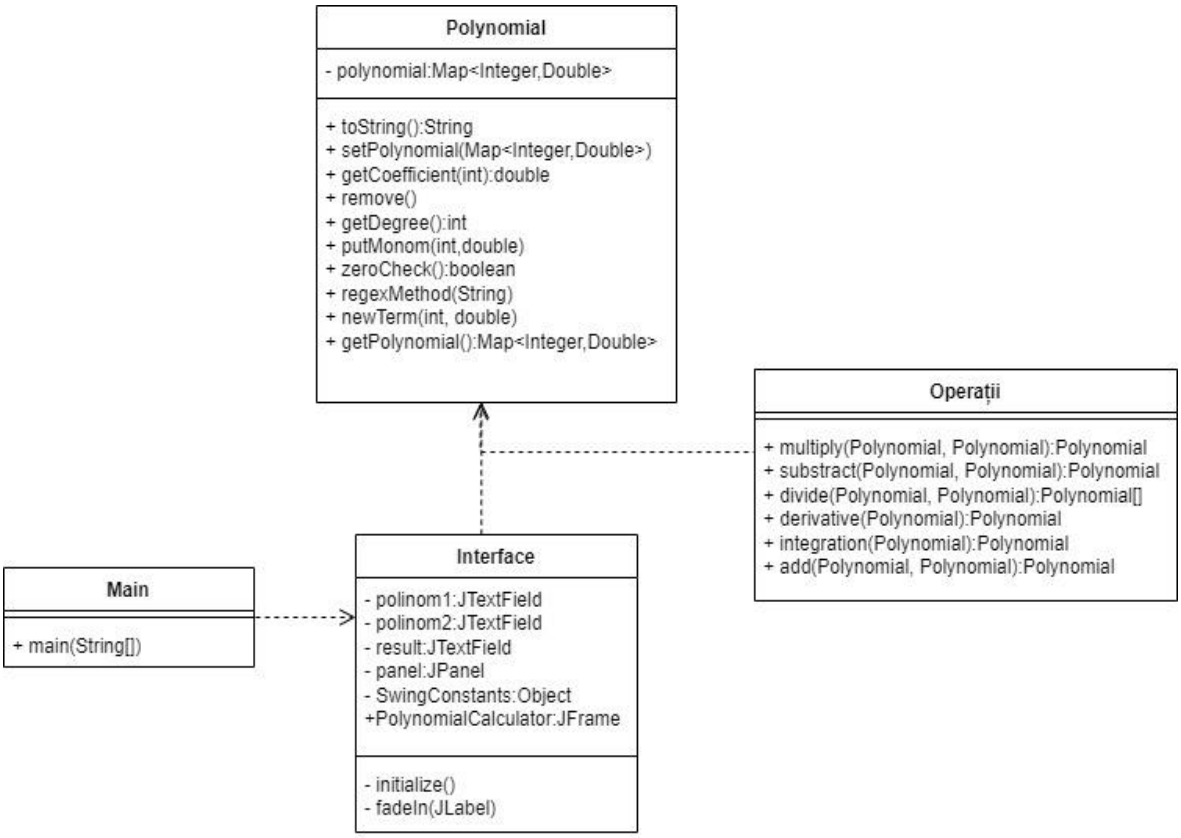
Clasa conține mai multe metode care efectuează operații cu polinoame, inclusiv adunare, scădere, înmulțire, derivare, integrare și împărțire. Fiecare metodă din cele enumerate primește două obiecte de tip Polynomial (polinoame) ca parametri și returnează un alt obiect Polynomial ca rezultat, iar derivarea și integrarea primesc un obiect de tip Polynomial. Metodele folosesc iterații prin elementele polinoamelor folosind iterații Map.Entry.

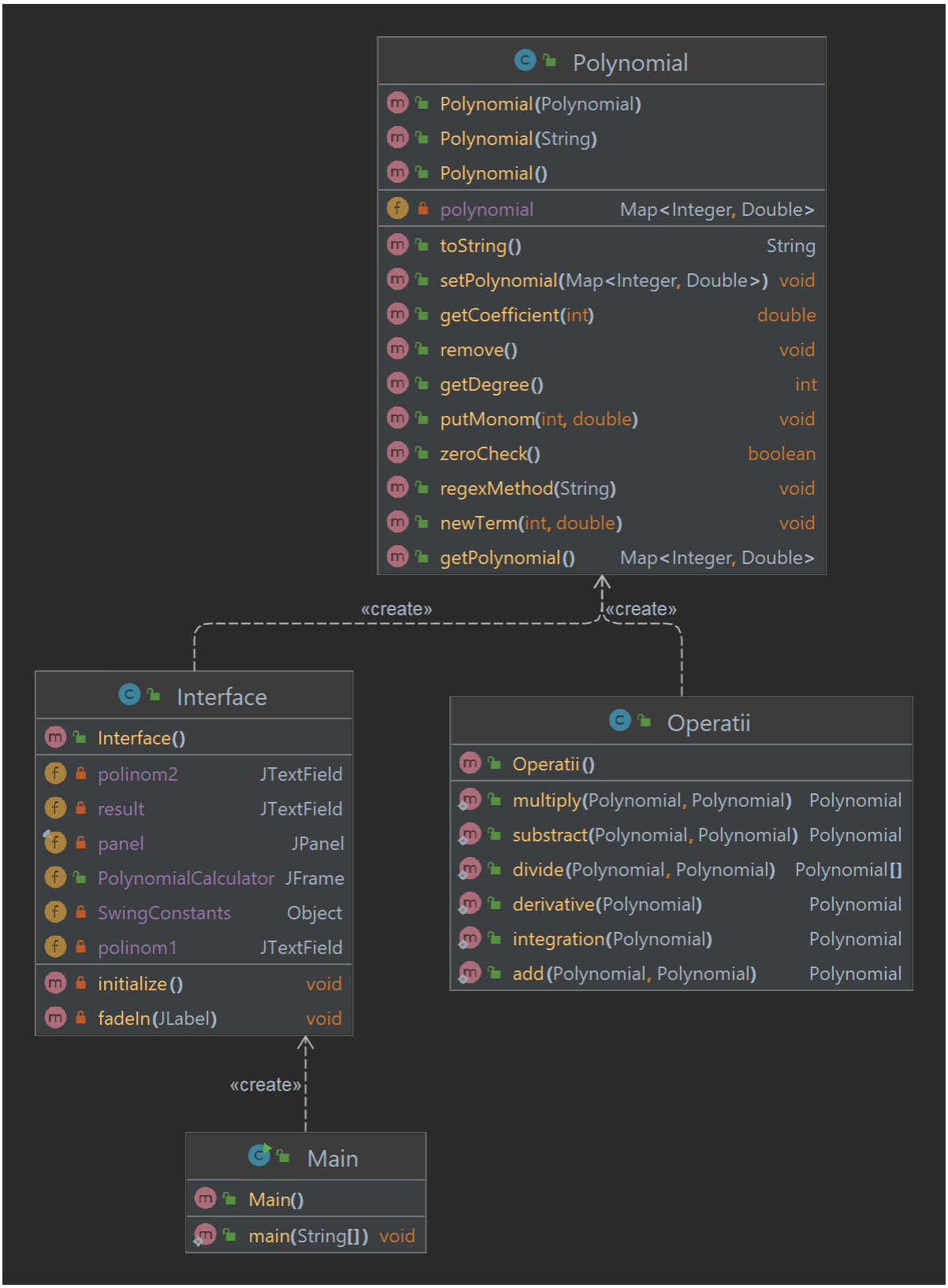
- Interface

Interfața grafică a fost implementată folosind biblioteca Java Swing. Codul folosește diverse clase și metode din bibliotecă pentru a crea interfața utilizatorului, a seta proprietățile diferitelor elemente și a gestiona intrarea utilizatorului.

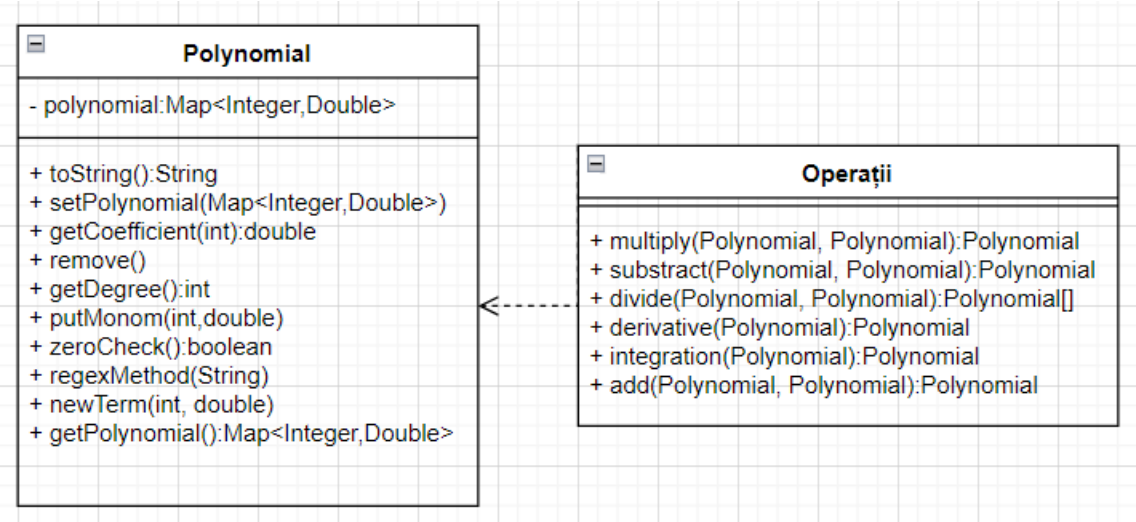
3.2) Diagrama UML

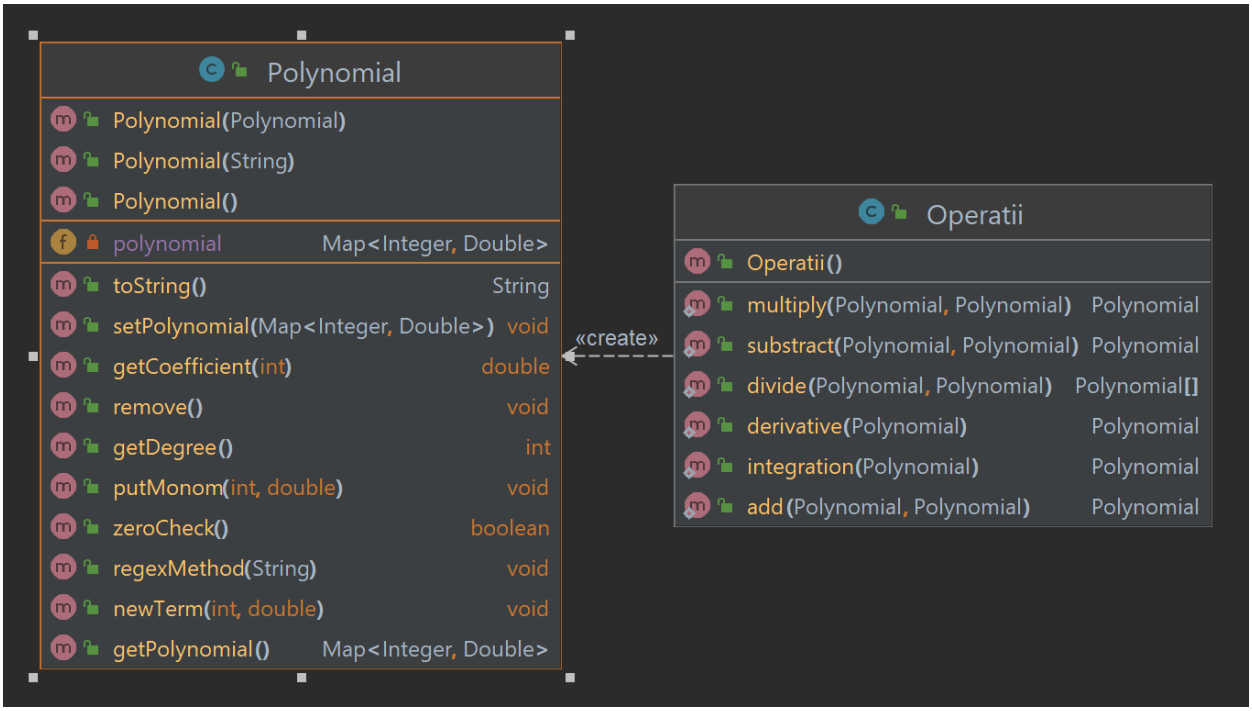
- Diagrama UML care reprezintă clasele și legăturile dintre ele



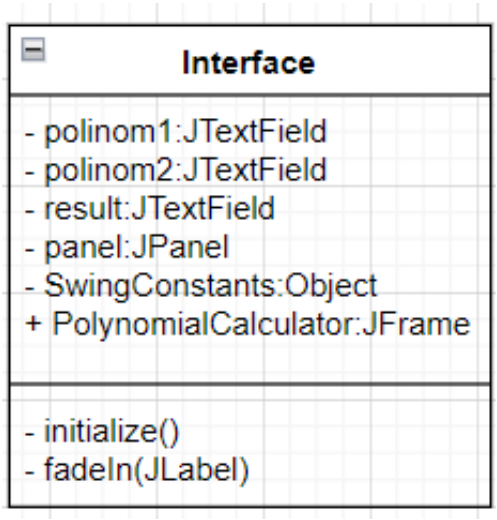

Polynomial
Polynomial(Polynomial)
Polynomial(String)
Polynomial()
polynomial Map<Integer, Double>
toString() String
setPolynomial(Map<Integer, Double>) void
getCoefficient(int) double
remove() void
getDegree() int
putMonom(int, double) void
zeroCheck() boolean
regexMethod(String) void
newTerm(int, double) void
getPolynomial() Map<Integer, Double>
«create»
«create»
Interface
Interface()
polinom2 JTextField
result JTextField
panel JPanel
PolynomialCalculator JFrame
SwingConstants Object
polinom1 JTextField
initialize() void
fadeIn(JLabel) void
Operatii
Operatii()
multiply(Polynomial, Polynomial) Polynomial
substract(Polynomial, Polynomial) Polynomial
divide(Polynomial, Polynomial) Polynomial[]
derivative(Polynomial) Polynomial
integration(Polynomial) Polynomial
add(Polynomial, Polynomial) Polynomial
«create»
Main
Main()
main(String[]) void

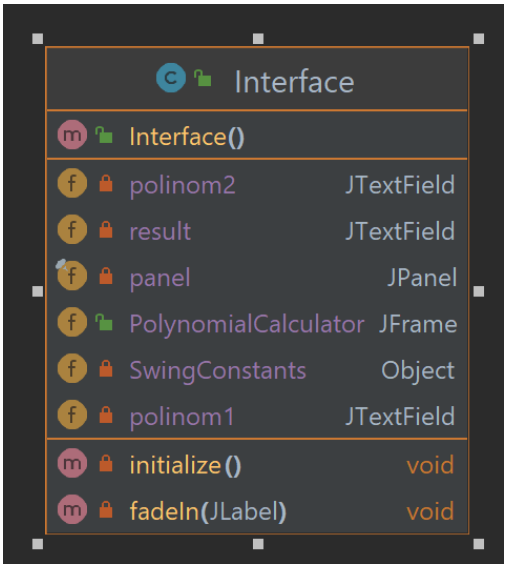
- Diagrama UML pentru pachetul operations





- Diagrama UML pentru pachetul gui





3.3) Structuri de date folosite

Ca structură de date am folosit Map, care este implementată prin intermediul clasei TreeMap, care asigură păstrarea perechilor sortate după cheie (exponent).

Alegerea Map are mai multe avantaje. În primul rând, permite accesul rapid la coeficienții pentru fiecare exponent. De asemenea, oferă o modalitate convenabilă de a itera prin fiecare termen din polinom, în ordine crescătoare a exponentului, deoarece TreeMap este utilizat pentru a stoca elementele.

4. Implementare

- **Polynomial**

```
public class Polynomial {
    25 usages
    private Map<Integer, Double> polynomial; // Map of exponent to coefficient
    19 usages  new *
    public Polynomial() {

        polynomial = new TreeMap<>();
    }
```

Câmpuri:

TreeMap<Integer, Double> polynomial: mapează exponentul la coeficientul corespunzător în polinom

Metode:

Constructorul fără argumente: Inițializează polinomul ca un obiect de tip "TreeMap" (o implementare a interfeței "Map" care ordonează elementele după cheie).

Constructorul cu argument de tip "Polynomial": Inițializează polinomul cu termenii unui alt obiect de tip "Polynomial".

putMonom(int degree, double coeff): Adaugă un nou termen la polinom, dat prin grad și coeficient. Dacă gradul există deja în polinom, se adaugă coeficientul la cel existent.

Constructorul cu argument de tip "String": Inițializează polinomul dintr-un șir de caractere. Acesta utilizează o expresie regulată pentru a identifica fiecare termen și apoi apelează metoda "putMonom" pentru a adăuga termenii la polinom.

getPolynomial(): Returnează polinomul ca un obiect de tip "Map<Integer, Double>".

setPolynomial(Map<Integer, Double> polynomial): Setează polinomul utilizând un obiect de tip "Map<Integer, Double>".

newTerm(int degree, double coefficient): Adaugă un nou termen la polinom, dat prin grad și coeficient. Dacă gradul există deja în polinom, se adaugă coeficientul la cel existent. Dacă coeficientul este 0, se elimină termenul din polinom.

toString(): Returnează polinomul ca un șir de caractere, în forma "a_n x^n + a_n-1 x^n-1 + ... + a_0", unde "a_n" este coeficientul termenului cu gradul cel mai mare.

zeroCheck(): Verifică dacă polinomul este egal cu 0 (adica fiecare coeficient este 0).

getCoefficient(int exponent): Returnează coeficientul termenului cu gradul dat. Dacă gradul nu există în polinom, se returnează 0.

getDegree(): Returnează gradul polinomului, adica gradul celui mai mare termen.

remove(): Elimină din polinom termenii cu coeficientul 0.

regexMethod(String polynom): Metodă privată care folosește o expresie regulată pentru a identifica termenii unui polinom dintr-un șir de caractere și apoi apelează metoda "putMonom" pentru a adăuga termenii la polinom. Această metodă este folosită de constructorul cu argument de tip "String".

- **Operații:**

Clasa "Operatii" este o clasă utilitară care conține mai multe metode statice pentru a efectua operații cu polinoame.

Metode:

add(Polynom, Polynom): adaugă două polinoame și returnează rezultatul. Această metodă parcurge fiecare termen din cele două polinoame și adaugă termenii cu aceeași putere.

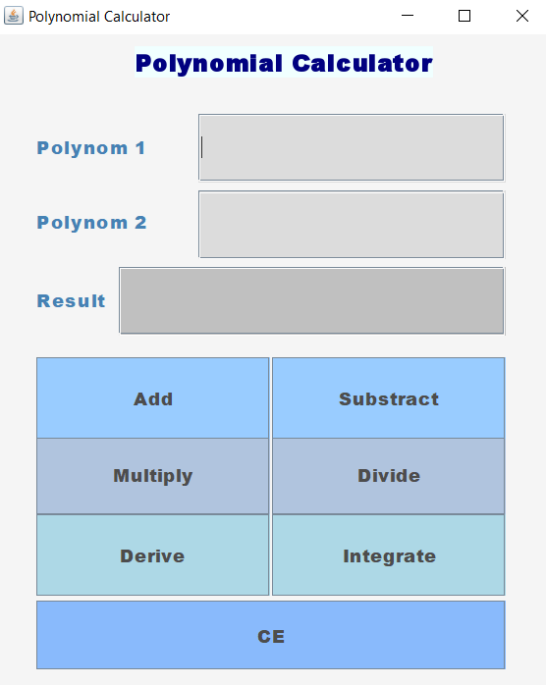
substract(Polynom, Polynom): scade un polinom din altul și returnează rezultatul. Această metodă parcurge fiecare termen din cele două polinoame și scade termenii cu aceeași putere.

multiply(Polynom, Polynom): înmulțește două polinoame și returnează rezultatul. Această metodă parcurge fiecare termen din primul polinom și îl înmulțește cu fiecare termen din al doilea polinom. Rezultatele sunt adăugate la un nou polinom, care este returnat.

derivative(Polynom): calculează derivata unui polinom și returnează rezultatul. Această metodă parcurge fiecare termen din polinom și calculează derivata termenului respectiv. Rezultatele sunt adăugate la un nou polinom, care este returnat.

integration(Polynom): calculează integrala unui polinom și returnează rezultatul. Această metodă parcurge fiecare termen din polinom și calculează integrala termenului respectiv. Rezultatele sunt adăugate la un nou polinom, care este returnat.

- **Interface:**



Aceasta este o clasă Java Swing GUI pentru un Calculator de Polinoame care primește două polinoame ca intrare și returnează rezultatul operației care se efectuează asupra acestora. GUI-ul include trei câmpuri text pentru polinoame și rezultat, precum și butoane pentru a efectua operații precum adunare, scădere, înmulțire, împărțire, derivare, integrare.

Interface utilizează biblioteca Java Swing pentru a crea o interfață grafică cu utilizatorul. Clasa include un obiect JFrame, mai multe obiecte JLabel și mai multe obiecte JButton. Interfața include, de asemenea, mai multe obiecte JTextField pentru introducerea datelor de intrare și afișarea rezultatelor.

Clasa include o metodă auxiliară privată numită fadeIn, care este utilizată pentru a crea o animație de estompare pentru eticheta titlului interfeței.

Funcționalitate:

- În câmpurile polinom1, polinom2 se introduc polinoamele corespunzătoare.
- Se selectează operația matematică dorită prin apăsarea unuia dintre butoanele: Add, Substract, Multiply, Divide, Derive, Integrate.
- Rezultatul va fi afișat în căsuța corespunzătoare.
- Pentru a șterge conținutul celor trei câmpuri se utilizează butonul CE.

- **Main**

În interiorul metodei main, se creează o instanță a clasei Interface, care este responsabilă pentru crearea și afișarea interfeței grafice a calculatorului de polinoame.

5. Rezultate

Pentru a verifica daca implementarea operatiilor cu polinoame este corecta, am scris teste pentru fiecare operatie din clasa Operatii folosind framework-ul JUnit. Am inclus aceste teste in clasa OperationTest si le-am rulat pentru a verifica daca operatiile returneaza rezultatele asteptate. In acest fel, am putut sa mă asigur ca implementarea operatiilor este corecta si ca aplicatia functioneaza corect pentru toate cazurile de utilizare.

- Adunarea

```
public class OperationTest {
    no usages  new *
    @Test
    public void addTest() throws Exception {
            Polynomial p1 = new Polynomial();
            p1.newTerm( degree: 3,  coefficient: 1);
            p1.newTerm( degree: 2,  coefficient: -2);
            p1.newTerm( degree: 1,  coefficient: 6);
            p1.newTerm( degree: 0,  coefficient: -5);

            Polynomial p2 = new Polynomial();
            p2.newTerm( degree: 2,  coefficient: 1);
            p2.newTerm( degree: 0,  coefficient: -1);

            Operatii op = new Operatii();
            Polynomial p3 = op.add(p1, p2);
            String op1_expected = "-6.0+6.0x-x^2+x^3";
            String str1 = p3.toString();
            assertEquals(op1_expected, str1);

    } }
```

Acesta este un exemplu de test pentru operatia de adunare a doua polinoame folosind JUnit. Testul creaza doua polinoame (p1 si p2), apoi foloseste clasa Operatii pentru a aduna cele doua polinoame si sa stocheze rezultatul in p3. Se asteapta ca rezultatul sa fie "-6.0+6.0x-x^2+x^3", care este stocat in variabila op1_expected.

Apoi, testul apeleaza metoda toString() a polinomului p3 pentru a obtine o reprezentare sub forma de sir de caractere a polinomului rezultat. Acest sir de caractere este stocat in variabila str1.

În cele din urma, testul utilizeaza metoda assertEquals() a JUnit pentru a verifica daca string-ul asteptat (op1_expected) este egal cu string-ul returnat (str1) de polinomul p3. Daca cele doua string-uri sunt egale, testul este considerat ca trecut cu succes.

- Scăderea

```
@Test
public void substractTest() {
        Polynomial p1 = new Polynomial();
        p1.newTerm( degree: 3,  coefficient: 1);
        p1.newTerm( degree: 2,  coefficient: -2);
        p1.newTerm( degree: 1,  coefficient: 6);
        p1.newTerm( degree: 0,  coefficient: -5);

        Polynomial p2 = new Polynomial();
        p2.newTerm( degree: 2,  coefficient: 1);
        p2.newTerm( degree: 0,  coefficient: -1);

        Operatii op = new Operatii();
        Polynomial p4 = op.substract(p1, p2);
        String op2_expected = "-4.0+6.0x-3.0x^2+x^3";
        String str2 = p4.toString();
        assertEquals(op2_expected, str2);

}
```

Acesta este un alt test din clasa de testare "OperationTest", care testează funcționarea corectă a metodei "substract" din clasa "Operatii". În această metodă de testare, se creează două obiecte "Polynomial", se adaugă diferiti termeni la acestea, apoi se apelează metoda "substract" din clasa "Operatii" pentru a obține diferența dintre cele două polinoame. Rezultatul este comparat cu un șir de caractere care conține polinomul așteptat ca rezultat. Acesta este un alt exemplu de utilizare a metodei "assertEquals" pentru a verifica dacă rezultatul așteptat și cel obținut sunt egale.

- Înmulțirea

```
    @Test
    public void multiplyTest() {
            Polynomial p1 = new Polynomial();
            p1.newTerm( degree: 3,  coefficient: 1);
            p1.newTerm( degree: 2,  coefficient: -2);
            p1.newTerm( degree: 1,  coefficient: 6);
            p1.newTerm( degree: 0,  coefficient: -5);

            Polynomial p2 = new Polynomial();
            p2.newTerm( degree: 2,  coefficient: 1);
            p2.newTerm( degree: 0,  coefficient: -1);

            Operatii op = new Operatii();
            Polynomial p5 = op.multiply(p1, p2);
            String op3_expected = "5.0-6.0x-3.0x^2+5.0x^3-2.0x^4+x^5";
            String str3 = p5.toString();
            assertEquals(op3_expected, str3);

    }
```

Acesta este un test pentru metoda de înmulțire implementată în clasa Operatii. Testul creează două polinoame și le înmulțește utilizând metoda multiply din clasa Operatii. Apoi, se compară șirul de caractere așteptat (op3_expected) cu șirul de caractere obținut prin transformarea în șir de caractere a rezultatului obținut prin multiplicare (str3). Dacă cele două șiruri sunt egale, testul este considerat trecut, în caz contrar va eșua.

- Împărțirea

```
@Test
public void divideTest() throws Exception {
        Polynomial p1 = new Polynomial();
        p1.newTerm( degree: 3,  coefficient: 1);
        p1.newTerm( degree: 2,  coefficient: -2);
        p1.newTerm( degree: 1,  coefficient: 6);
        p1.newTerm( degree: 0,  coefficient: -5);

        Polynomial p2 = new Polynomial();
        p2.newTerm( degree: 2,  coefficient: 1);
        p2.newTerm( degree: 0,  coefficient: -1);

        Operatii op = new Operatii();
        Polynomial[] p8 = op.divide(p1, p2);
        String op6_expected = "-2.0+x";
        String op7_expected = "-7.0+7.0x";

         String str6 = p8[0].toString();
         String str7 = p8[1].toString();
         assertEquals(op6_expected, str6);
         assertEquals(op7_expected, str7);

}
```

Acesta este un test pentru operația de împărțire a două polinoame. Sunt create două polinoame, p1 și p2, iar apoi se aplică operația de împărțire folosind clasa Operații Rezultatul împărțirii este un vector cu două elemente, p8[0] și p8[1], care reprezintă câtul și restul împărțirii.

În acest test, se verifică dacă împărțirea polinomului p1 la polinomul p2 produce rezultatul corect. Se așteaptă ca câtul să fie "-2.0+x" și restul să fie "-7.0+7.0x". Aceste valori sunt stocate în op6_expected și op7_expected, iar rezultatul real este obținut prin conversia polinoamelor în șiruri de caractere (String) folosind metoda toString() și comparând apoi aceste șiruri cu valorile așteptate folosind metoda assertEquals().

- Derivarea

```
@Test
public void deriveTest() throws Exception {
        Polynomial p1 = new Polynomial();
        p1.newTerm( degree: 3,  coefficient: 1);
        p1.newTerm( degree: 2,  coefficient: -2);
        p1.newTerm( degree: 1,  coefficient: 6);
        p1.newTerm( degree: 0,  coefficient: -5);

        Operatii op = new Operatii();
        Polynomial p6 = op.derivative(p1);
        String op4_expected = "6.0-4.0x+3.0x^2";
        String str4 = p6.toString();
        assertEquals(op4_expected, str4);
}
```

Această testare verifică dacă funcția derivative din clasa Operatii calculează derivata polinomului corect. Polinomul de test are termenii: 3x^3 - 2x^2 + 6x - 5. Se așteaptă ca derivata să fie: 9x^2 - 4x + 6. Se verifică dacă polinomul obținut prin apelarea metodei derivative este identic cu polinomul așteptat prin apelarea metodei assertEquals.

- Integrarea

```
@Test
public void integrateTest() {
        Polynomial p1 = new Polynomial();
        p1.newTerm( degree: 3,  coefficient: 1);
        p1.newTerm( degree: 2,  coefficient: -2);
        p1.newTerm( degree: 1,  coefficient: 6);
        p1.newTerm( degree: 0,  coefficient: -5);

        Operatii op = new Operatii();
        Polynomial p7 = op.integration(p1);
        String op5_expected = "5.0x+3.0x^2+0.67x^3+0.25x^4";
        String str5 = p7.toString();
        assertEquals(op5_expected, str5);
}
```

Această testare verifică dacă metoda integration a clasei Operatii returnează polinomul integrat corect pentru un polinom dat ca input. Se creează un polinom p1 și se așteaptă ca rezultatul metodei integration să fie polinomul integrat corect, reprezentat de șirul de caractere

op5_expected. Se verifică dacă șirul de caractere returnat de metoda toString a obiectului Polynomial este egal cu op5_expected.

După ce am rulat testele, se poate observa că toate au fost corecte.

```
[INFO] Results:
[INFO]
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.698 s
[INFO] Finished at: 2023-03-28T02:32:35+03:00
[INFO] ------------------------------------------------------------------------

Process finished with exit code 0
```

6. Concluzii

În urma acestei teme, am reușit să înțeleg conceptul de polinoame și operațiile matematice asociate acestora, cum ar fi adunarea, scăderea, înmulțirea, împărțirea, derivarea și integrarea. Am avut o experiență practică în implementarea acestor operații în limbajul Java, folosind clasa Polynomial. Totodată, am învățat să folosesc JUnit pentru a implementa testele unitare necesare verificării corectitudinii programului. Această temă mi-a dezvoltat abilitățile de rezolvare a problemelor și de gândire algoritmică.

Posibile dezvoltari ulterioare includ:

- Adăugarea de functionalitati noi, cum ar fi extragerea radacinilor sau evaluarea polinoamelor intr-un anumit punct
- Extinderea implementarii pentru a permite lucrul cu polinoame cu coeficienti rationali, nu doar intregi si reali.
- Adăugarea unei interfete grafice mai elaborate, care sa ofere utilizatorului mai multe optiuni si sa ofere o experienta mai placuta la utilizare.

7. Bibliografie

1. https://dsrl.eu/courses/pt/materials/PT2023_A1_S1.pdf
2. https://dsrl.eu/courses/pt/materials/PT2023_A1_S3.pdf
3. https://stackoverflow.com/

4. https://www.javatpoint.com/java-swing
5. *https://regexr.com*