

DOCUMENTATIE

TEMA 1

NUME STUDENT: PEPELEA IOANA-ADINA

GRUPA: 30228

CUPRINS

1.	Obiectivul temei	3
2.	Analiza problemei, modelare, scenarii, cazuri de utilizare	3
3.	Proiectare.....	3
4.	Implementare	3
5.	Rezultate.....	3
6.	Concluzii	3
7.	Bibliografie.....	3

1. Obiectivul temei

Obiectivul principal al temei este realizarea unui calculator de polinoame, care sa fie capabil sa execute operatii, cum ar fi: adunarea, scaderea, inmultirea, impartirea, precum si derivarea si integrarea unui polinom. Obiectivele secundare ale temei sunt urmatoarele:

- Realizarea unei interfete grafice prin care utilizatorul sa poata interactiona cu programul;
- Extragerea input-ului introdus de utilizator de la tastatura;
- Validarea input-ului si emiterea unui mesaj de eroare in cazul in care acesta nu corespunde unui polinom valid;
- Efectuarea operatiei alese de utilizator cu ajutorul unor butoane;
- Afisarea rezultatului in cadrul interfetei grafice.

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

Analizand cerinta temei, se poate observa ca aceasta poate fi descompusa in mai multe subprobleme de dimensiuni mai mici, pe care le putem modela cu usurinta: crearea unui polinom pe baza input-ului introdus de un utilizator in interfata grafica, validarea acestuia, efectuarea operatiilor si afisarea rezultatului. Rezolvarea acestor subprobleme presupune existenta mai multor clase, cu attribute si metode specifice, care modeleaza conceptele reale utilizate in rezolvarea operatiilor cu polinoame. Totodata, utilizarea aplicatiei de catre un utilizator din lumea reala da nastere mai multor scenarii care pot avea loc, in functie de modul in care utilizatorul interactioneaza cu interfata grafica a aplicatiei si de operatiile pe care acesta doreste sa le efectueze asupra polinoamelor. Aceste cazuri de utilizare pot fi descrise cu ajutorul unor documente user-case, in care e descrisa evolutia sistemului si interactiunea utilizatorului cu acesta.

1. **Use case:** Efectuarea operatiilor care au ca input 2 polinoame.

Primary actor: User

Main success scenario:

- 1.1. User-ul introduce cele doua polinoame, fiecare in text box-ul corespunzator.
- 1.2. User-ul alege operatia pe care doreste sa o efectueze asupra polinoamelor, apasand unul din cele patru butoane: Add, Subtract, Multiply, Divide.

- 1.3. Se verifica daca cele doua polinoame sunt valide.
 - 1.4. Se obtin cele doua polinoame, plecand de la cele doua string-uri introduse de utilizator.
 - 1.5. Se efectueaza operatia aleasa de utilizator.
 - 1.6. Daca operatia aleasa a fost una dintre urmatoarele: adunarea, scaderea, inmultirea, se afiseaza rezultatul intr-un singur label, in dreptul label-ului cu textul "Result:".
- Daca operatia aleasa a fost impartirea, rezultatul se afiseaza in dreptul label-ului cu textul "Result:" si restul se afiseaza in dreptul label-ului cu textul "Rest:".

Alternative sequence:

User-ul introduce unul sau ambele input-uri invalide. Se afiseaza un mesaj de eroare, iar scenariul revine la pasul 1.

2. **User case:** Efectuarea operatiilor care au ca input un singur polinom.

Primary actor: User.

Main success scenario:

- 2.1. User-ul poate introduce doar un singur polinom, in text box-ul corespunzator.
- 2.2. User-ul alege operatia pe care doreste sa o efectueze, pentru primul polinom din interfata grafica, apasand unul din cele 2 butoane: Derivate, Integrate.
- 2.3. Se verifica daca polinomul ales este valida.
- 2.4. Se obtine polinomul, plecand de la string-ul introdus de utilizator.
- 2.5. Se efectueaza operatia aleasa pe acest polinom.
- 2.6. Se afiseaza polinomul rezultat, in acelasi label ca in cazul operatiilor pe doua polinoame.

Alternative sequence:

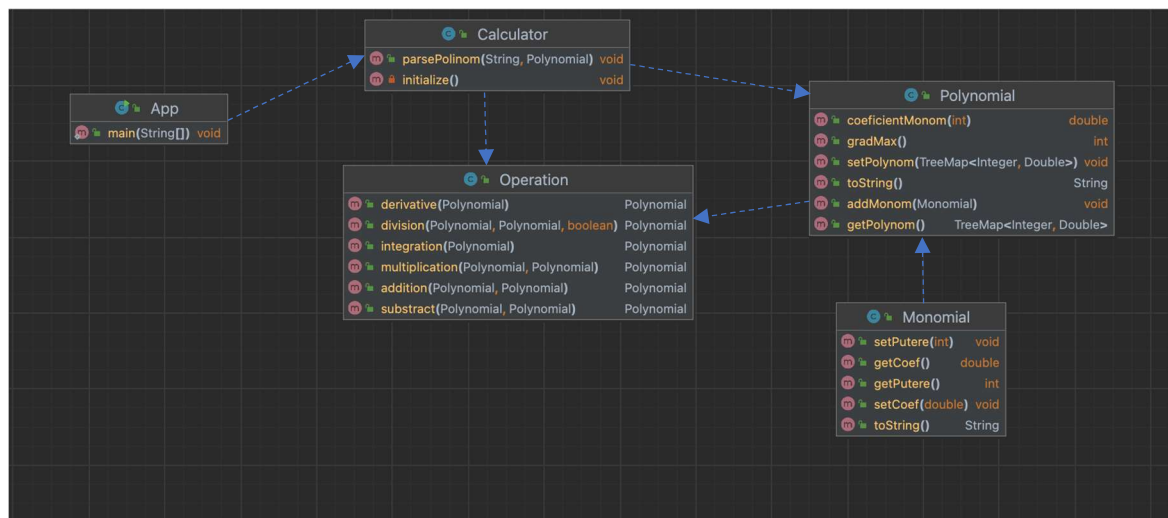
Polinomul asupra caruia s-a ales efectuarea operatiei este invalid. Se afiseaza un mesaj de eroare, iar scenariul revine la pasul 1.

3. Proiectare

Aplicatia este proiectata conform conceptelor de baza OOP. Abstractizarea presupune ca fiecare obiect are un rol bine definit, putand sa comunice cu celelalte obiecte, sa primeasca si sa furnizeze informatii, fara a da detalii despre implementarea facilitatilor. Astfel, sistemul este compus din patru clase, fiecare avand un rol bine stabilit: Monomial, Polynomial, Operation,

Calculator. Incapsularea asigura ca obiectele nu pot schimba starea interna a altor obiecte decat prin metode de tip setter, toate atribuirile clasei avand modificatorul de acces private.

Relatiile dintre clasele definite in cadrul sistemului este cuprinsa in diagrama UML de mai jos:



4. Implementare

4.1. Clasa Monomial

Aceasta clasa modeleaza un monom, avand ca attribute private gradul(putere) de tip `int` si coeficientul(coef) de tip `double`. Clasa contine metodele clasice de tip setter, getter si `toString()`.

4.2. Clasa Polynomial

In general, un polinom este alcatuit dintr-o variabila polynom de tipul `TreeMap`, care asociaza fiecarei puteri(adica cheia), un coeficient(adica valoarea).

Metoda `addMonom(Monomial)` adauga un nou monom in polinom. In cadrul metodei suprascrise `toString()`, se parcurge `TreeMap`-ul de monoame, apeland metoda `toString()` pentru fiecare monom din polinom, la final returnandu-se un `String` care ia forma unui polinom. In aceasta clasa am creat doua alte metode, una fiind numita `gradMax()`, care returneaza gradul maxim al unui polinom si si metoda `coefficientMonom(i)`, care returneaza coeficientul monomului care are puterea cu numarul parametrului pe care il dam functiei.

4.3. Clasa Operation

In aceasta clasa sunt definite metodele pentru efectuarea operatiilor care vor fi efectuate pe polinoame: adunare, scadere, inmultire, impartire, precum si derivarea si integrarea unui polinom.

Metoda `addition()` primește ca parametri două polinoame, returnând polinomul rezultat în urma adunării celor două polinoame. Adunarea celor două polinoame se realizează prin parcurgerea celor două polinoame, astfel:

```
public Polynomial addition(Polynomial p1, Polynomial p2) {
    Polynomial polinomRez = new Polynomial();
    for (Map.Entry<Integer, Double> monom1 : p1.getPolynom().entrySet()) {
        for (Map.Entry<Integer, Double> monom2 : p2.getPolynom().entrySet()) {
            if (monom1.getKey() == monom2.getKey()) {
                polinomRez.addMonom(new Monomial(monom1.getKey(), coef: monom1.getValue() + monom2.getValue()));
            }
        }
    }
    for (Map.Entry<Integer, Double> monom1 : p1.getPolynom().entrySet()) {
        if (polinomRez.getPolynom().containsKey(monom1.getKey()) == false) {
            polinomRez.addMonom(new Monomial(monom1.getKey(), monom1.getValue()));
        }
    }
    for (Map.Entry<Integer, Double> monom2 : p2.getPolynom().entrySet()) {
        if (polinomRez.getPolynom().containsKey(monom2.getKey()) == false) {
            polinomRez.addMonom(new Monomial(monom2.getKey(), monom2.getValue()));
        }
    }
    return polinomRez;
}
```

Dacă gradele celor două monoame sunt egale, coeficienții lor se adună și gradul se păstrează, astfel noul monom se adaugă într-un polinom rezultat (`polinomRez`). La final, în cazul în care polinoamele nu au fost parcurse până la final, acestea sunt parcurse, iar monoamele rămase sunt adăugate la polinomul rezultat.

Metoda `subtract()`, care realizează scăderea, este foarte asemănătoare cu metoda de adunare, singura diferență fiind scăderea coeficienților, în locul adunării lor:

```
public Polynomial subtract(Polynomial p1, Polynomial p2) {
    Polynomial polinomRez = new Polynomial();
    for (Map.Entry<Integer, Double> monom1 : p1.getPolynom().entrySet()) {
        for (Map.Entry<Integer, Double> monom2 : p2.getPolynom().entrySet()) {
            if (monom1.getKey() == monom2.getKey()) {
                polinomRez.addMonom(new Monomial(monom1.getKey(), coef: monom1.getValue() - monom2.getValue()));
            }
        }
    }
    for (Map.Entry<Integer, Double> monom1 : p1.getPolynom().entrySet()) {
        if (polinomRez.getPolynom().containsKey(monom1.getKey()) == false) {
            polinomRez.addMonom(new Monomial(monom1.getKey(), monom1.getValue()));
        }
    }
    for (Map.Entry<Integer, Double> monom2 : p2.getPolynom().entrySet()) {
        if (polinomRez.getPolynom().containsKey(monom2.getKey()) == false) {
            polinomRez.addMonom(new Monomial(monom2.getKey(), -monom2.getValue()));
        }
    }
    return polinomRez;
}
```

În cadrul metodei `multiplication()`, care execută înmulțirea polinoamelor, se parcurg din nou polinoamele, folosind două `for-uri` imbricate, care vor înmulți fiecare monom al primului polinom cu fiecare monom al celui de al doilea polinom, astfel:

```

public Polynomial multiplication(Polynomial p1, Polynomial p2) {
    Polynomial polinomRez = new Polynomial();
    for (Map.Entry<Integer, Double> monom1 : p1.getPolynom().entrySet()) {
        for (Map.Entry<Integer, Double> monom2 : p2.getPolynom().entrySet()) {
            //polinomRez.addMonom(new Monomial(monom1.getKey()+monom2.getKey(), monom1.getValue() * monom2.getValue()));
            Monomial monomRez = new Monomial();
            monomRez.setCoef(monom1.getValue() * monom2.getValue());
            monomRez.setPutere(monom1.getKey() + monom2.getKey());
            if (polinomRez.getPolynom().containsKey(monomRez.getPutere()) == true) {
                monomRez.setCoef(monomRez.getCoef() + polinomRez.getPolynom().get(monomRez.getPutere()));
            }
            polinomRez.addMonom(monomRez);
        }
    }
    return polinomRez;
}

```

Metoda derivative(polynomial) returneaza polinomul derivat, al polinomului primit ca parametru. Se parcurge polinomul si derivarea se realizeaza astfel:

```

public Polynomial derivative(Polynomial p1) {
    Polynomial polinomRez = new Polynomial();
    for (Map.Entry<Integer, Double> monom1 : p1.getPolynom().entrySet()) {
        Monomial monomRez = new Monomial();
        monomRez.setCoef(monom1.getKey() * monom1.getValue());
        monomRez.setPutere(monom1.getKey() - 1);
        polinomRez.addMonom(monomRez);
    }
    return polinomRez;
}

```

Integrarea se realizeaza asemanator cu derivarea, formula de integrare inlocuind formula de derivare:

```

public Polynomial integration(Polynomial p1) {
    Polynomial polinomRez = new Polynomial();
    for (Map.Entry<Integer, Double> monom1 : p1.getPolynom().entrySet()) {
        Monomial monomRez = new Monomial();
        monomRez.setCoef(monom1.getValue() / (monom1.getKey() + 1));
        monomRez.setPutere(monom1.getKey() + 1);
        polinomRez.addMonom(monomRez);
    }
    return polinomRez;
}

```

Impartirea polinoamelor se realizeaza astfel:

- Se afla gradul maxim al celor doua polinoame pe care trebuie sa le impartim, iar cat timp gradul celui de al primului este mai mare decat gradul celui de al doilea vom face urmasorii pasi:
- Ne luam un nou monom, care are ca putere diferenta dintre gradul maxim al primului polinom si gradul maxim al celui de al doilea;
- Adaugam noul monom intr-un polinom, numit "result", care va fi rezultatul impartirii

-Intr-un polinom auxiliar, vom face inmultirea dintre noul nostru monom si fiecare monom din polinomul al doilea;

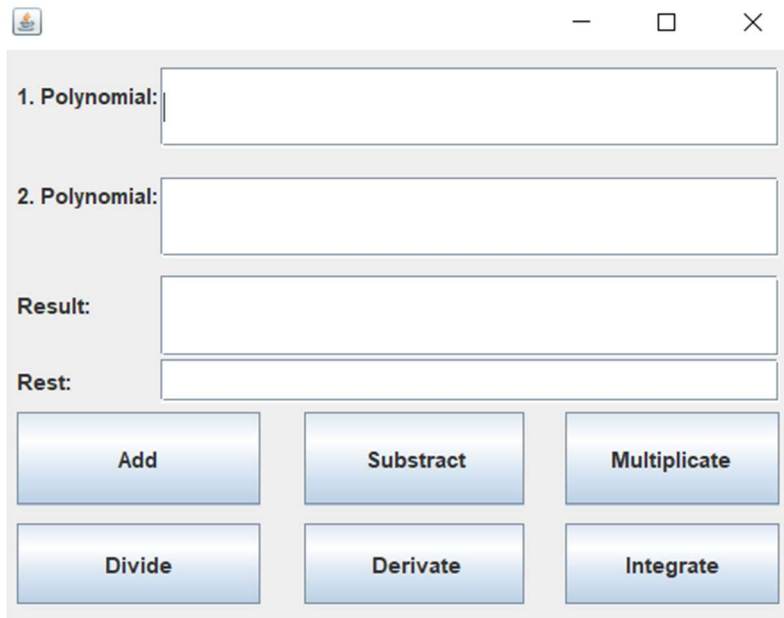
-Restul este reprezentat de adunarea dintre primul polinom si cel auxiliar.

```
public Polynomial division(Polynomial p1, Polynomial p2, boolean select) {
    Polynomial result=new Polynomial();
    Polynomial rest=new Polynomial();
    Polynomial polinomAux=new Polynomial();
    int gradp1= p1.gradMax();
    int gradp2= p2.gradMax();
    while(gradp1>=gradp2){
        polinomAux.getPolynom().clear();
        int putereDivizor=gradp1-gradp2;
        double coefDivizor=p1.coeficientMonom(gradp1)/p2.coeficientMonom(gradp2);
        Monomial monomAux = new Monomial();
        monomAux.setCoef(coefDivizor);
        monomAux.setPutere(putereDivizor);
        result.addMonom(monomAux);
        for(Map.Entry<Integer, Double> monom2 : p2.getPolynom().entrySet()) {
            Monomial monomAux1=new Monomial();
            monomAux1.setCoef(-(monom2.getValue()*coefDivizor));
            monomAux1.setPutere(monom2.getKey()+putereDivizor);
            polinomAux.addMonom(monomAux1);
        }
        Operation operatie1=new Operation();
        rest=operatie1.addition(p1, polinomAux);
        p1=rest;
        gradp1--;
    }
    if(select==true) return result;
    else return rest;
}
```

4.4. Clasa Calculator

Aceasta clasa implementeaza interfata grafica, cu care interactioneaza utilizatorul. In cadrul clasei sunt definite componente din Java Swing, printre care se afla label-uri, text box-uri si butoane. Tot aici, sunt definite metode prin care se extrage textul din cadrul text box-ului si se verifica ce buton este apasat de catre utilizator, pentru a se decide ce operatie sa se efectueze.

Interfata utilizator finala arata ca in figura de mai jos, avand un aspect simplist, dar intuitiv:



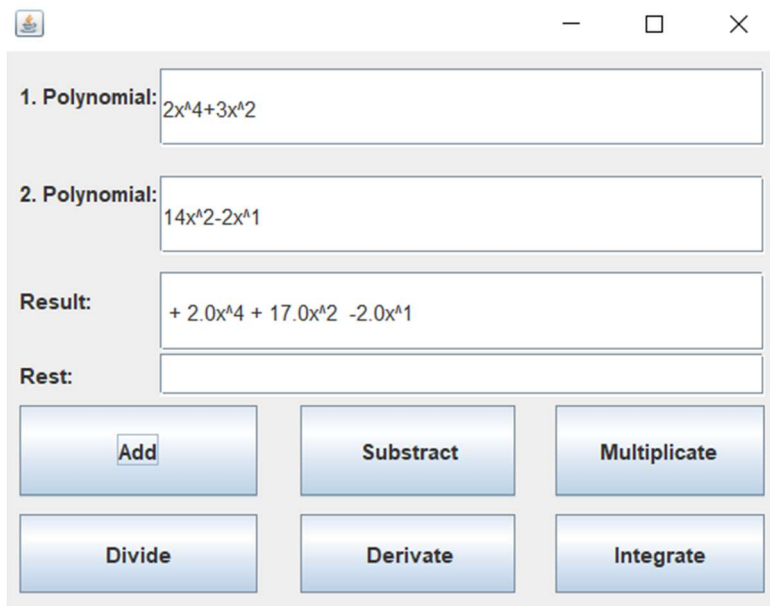
1. Polynomial:

2. Polynomial:

Result:

Rest:

Add	Subtract	Multiply
Divide	Derivate	Integrate



1. Polynomial:

2. Polynomial:

Result:

Rest:

Add	Subtract	Multiply
Divide	Derivate	Integrate

Tot in aceasta clasa am realizat metoda `parsePolinom(String input, Polynomial polinom)`, care verifica corectitudinea String-ului introdus de utilizator. Regex-ul utilizat incearca sa cuprinda toate cazurile in care un polinom este considerat valid: `"(-?\d+)[x]\^(-?\d+)"`.

5. Rezultate

In cadrul clasei de test, exista cate o metoda de test pentru fiecare operatie implementata asupra polinoamelor. Ca si scenariul de test, am considerat doua polinoame, cu ajutorul carora am testat fiecare operatie, pe rand, testand daca rezultatul returnat de metodele din clasa `Operation` este egal cu rezultatul asteptat, care a fost calculat in prealabil, cu ajutorul metodei

assertEquals()). Un exemplu de astfel de metoda de test este prezentata mai jos, celelalte fiind foarte asemanatoare cu acesta, apelul functiei si rezultatul fiind singurele diferite.

```
@Test
public void additionTest(){
    Monomial monom1=new Monomial( putere: 4, coef: 2);
    Monomial monom2=new Monomial( putere: 2, coef: 1);
    Monomial monom3=new Monomial( putere: 1, coef: 1);
    Monomial monom4=new Monomial( putere: 0, coef: -3);
    Polynomial polinom1=new Polynomial();
    polinom1.addMonom(monom1);
    polinom1.addMonom(monom2);
    polinom1.addMonom(monom3);
    polinom1.addMonom(monom4);
    Monomial monom5=new Monomial( putere: 2, coef: 1);
    Monomial monom6=new Monomial( putere: 1, coef: -4);
    Polynomial polinom2=new Polynomial();
    polinom2.addMonom(monom5);
    polinom2.addMonom(monom6);
    Operation operatie1=new Operation();
    Polynomial rezultat=operatie1.addition(polinom1, polinom2);
    assertEquals( expected: " -3.0x^0 -3.0x^1 + 2.0x^2 + 2.0x^4", rezultat.toString());
}
```

6. Concluzii

Calculatorul de polinoame implementat realizeaza operatiile de baza care se pot efectua asupra acestora, avand o interfata grafica intuitiva, usor de utilizat. Totusi, ca in cazul oricarei aplicatii, acestuia I se pot face imbunatatiri, atat la nivel de interfata grafica, prin adaugarea unor butoane care sa reprezinte numerele, ca in cazul unui calculator real, dar si la nivel de complexitate a operatiilor pe care le executa, cum ar fi adaugarea unei functionalitati care sa calculeze radacinile unui polinom.

7. Bibliografie

<https://courses.lumenlearning.com/beginalgebra/chapter/4-2-2-adding-and-subtracting-polynomials/>

https://dsrl.eu/courses/pt/materials/PT2023_A1_S1.pdf

https://dsrl.eu/courses/pt/materials/PT2023_A1_S2.pdf

https://dsrl.eu/courses/pt/materials/PT2023_A1_S3.pdf