DOCUMENTAȚIE

TEMA 1 : CALCULATOR POLINOMIAL

NUME STUDENT: Tarța Antonia-Maria
GRUPA: 30229

CUPRINS

1. Obiectivul temei

Obiectivul principal al acestei teme este proiectarea și implementarea unui calculator polinomial care dispune de o interfață grafică. Interfața prietenoasă a acestui calculator permite utilizatorului să introducă de la tastatură două polinoame asupra cărora se vor efectua operații matematice precum: adunare, scădere, înmulțire, împărțire, integrare și derivare, ulterior rezultatul va putea fi vizualizat.

Obiectivele secundare ale acestei teme sunt:

- Implementarea funcționalităților matematice: se vor implementa funcționalitățile matematice necesare calculatorului. Acest obiectiv va fi detaliat în capitolul „Implementare”.
- Proiectarea interfeței grafice: presupune realizarea unei interfețe grafice intuitivă și ușor de utilizat care să permită introducerea polinoamelor și selectarea operației matematice. Acest obiectiv va fi detaliat în capitolul „Proiectare”.
- Validarea datelor de intrare: datele de intrare introduse de utilizator trebuie să fie valide, altfel operația dorită nu va fi efectuată. Acest obiectiv va fi detaliat în capitolul de „Implementare”.
- Testarea aplicației: efectuarea testelor pentru a verifica dacă aplicația funcționează corect și dacă răspunsurile furnizate sunt cele așteptate. Acest obiectiv va fi detaliat în capitolul „Rezultate”.

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

Analiza problemei

În această aplicație se consideră polinoamele de o singură variabilă:

$$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Coeficienții sunt a_0, a_1,.., a_n. Coeficienții polinoamelor introduse de utilizator sunt numere întregi. Polinomul este alcătuit din monoame care sunt reprezentate de o expresie de forma „coeficient * x^p”, unde p este un numar natural.

Cerințe funcționale:

- Calculatorul ar trebui să permită utilizatorului să introducă două polinoame.
- Calculatorul ar trebui să permită utilizatorului să selecteze operația dorită.
- 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.
- Calculatorul ar trebui să permită utilizatorului să acceseze meniul „Help”.
- Calculatorul ar trebui să permită utilizatorului să reseteze valorile introduse.

Cerințe non-funcționale:

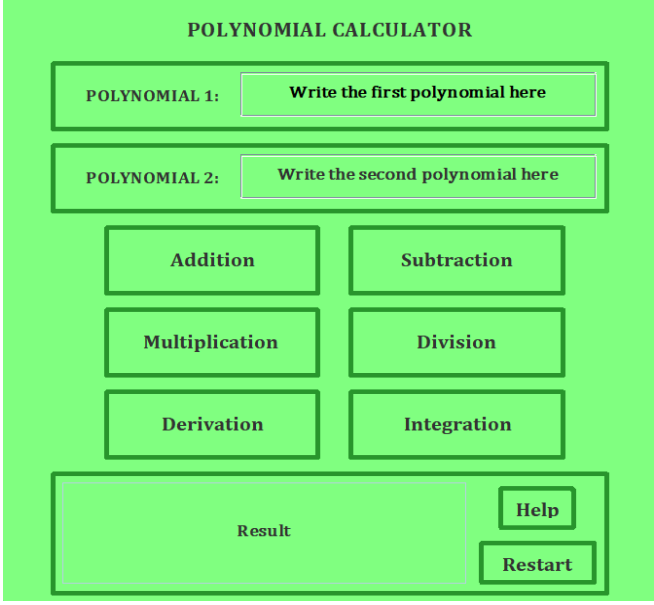
- Calculatorul de polinoame ar trebui să fie ușor de folosit
- Calculatorul de polinoame ar trebui să aibă o interfață prietenoasă.
- Calculatorul de polinoame ar trebui să afișeze mesaje de eroare în cazul în care utilizatorul nu introduce corespunzător polinoamele.
- Calculatorul de polinoame ar trebui să fie precis și să efectueze corect operațiile matematice.
- Aplicația trebuie să fie eficientă din punct de vedere al timpului de execuție și al memoriei utilizate.

Modelarea problemei

Utilizatorul va trebui să introducă două polinoame direct in interfață, în spatiul rezervat, iar daca datele introduse nu respectă formatul unui polinom, atunci se va afișa un mesaj de eroare. Acesta va putea consulta meniul „Help” și va putea să resete interfața. De asemenea, utilizatorul poate să selecteze operația dorită. Rezultatul operatiei va fi afisat.

Scenarii și cazuri de utilizare

Cazurile de utilizare și scenariile sunt în strânsă legătură cu acțiunile utilizatorului. Interfața îi permite acestuia să îmbine utilul cu plăcutul deoarece folosește o aplicație cu care este ușor de interacționat pentru a obține rezultatele dorite. Pentru o mai bună înțelegere a scenariului o să atașez o imagine cu interfața:

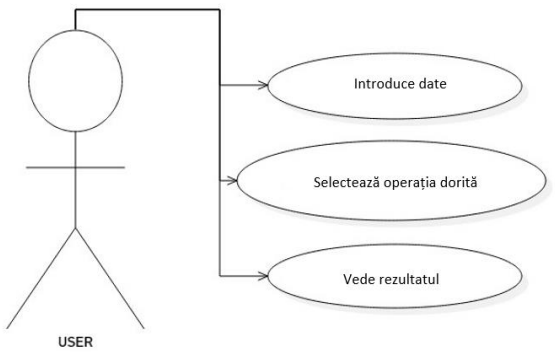


Scenariul principal:

1) Utilizatorul introduce cele două polinoame.
2) Utilizatorul selectează operația dorită.
3) Rezultatul este afișat.

Scenariul secundar:

- Utilizatorul introduce date invalide.
- Opțional se consultă meniul „Help” și se resetează căsuțele de text, se revine la pasul 1.



3. Proiectare

Proiectarea POO

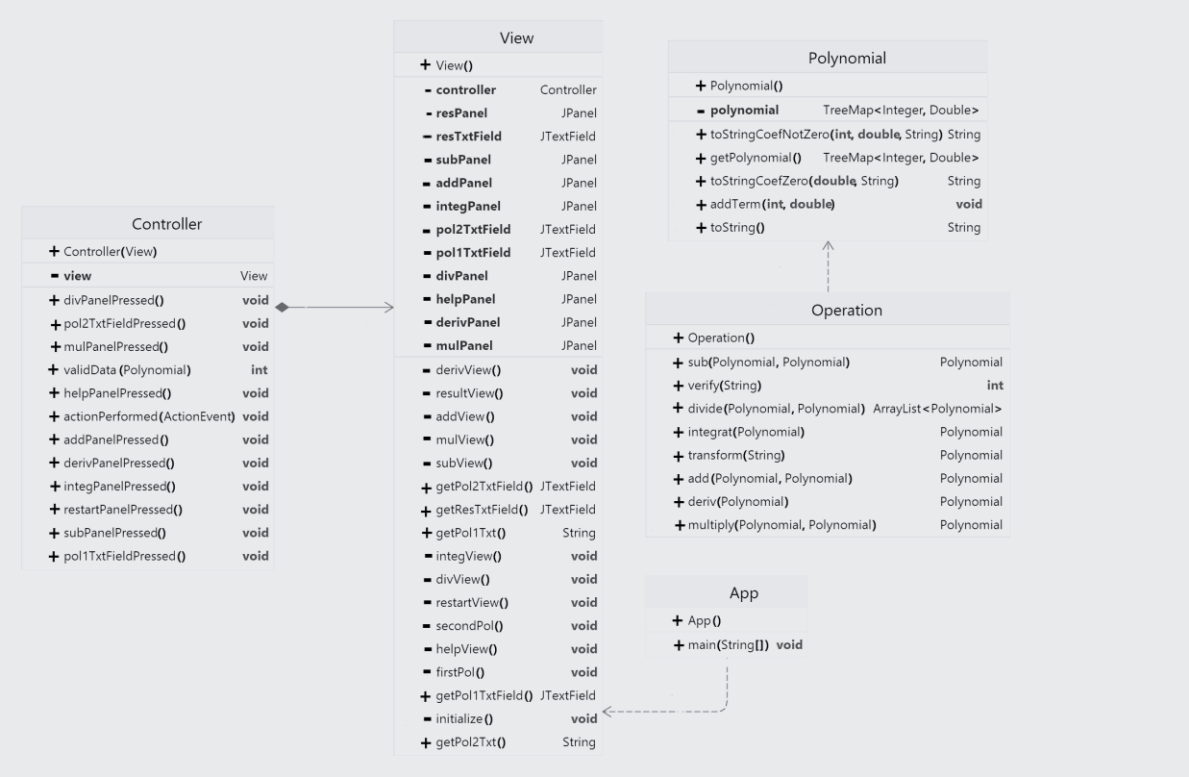
Aplicația conține 6 clase :

- App (MainClass)
- AppTest
- Polynomial
- Operation
- View
- Controller

Pachete :

- BusinessLogic(conține logica din spatele operațiilor matematice - clasa Operation)
- DataModels (conține clasa unde sunt modelate datele – clasa Polynomial)
- GUI (conține clasele care se ocupă de interfața grafică – View, Controller)

Diagrama UML



Structuri de date

Am folosit Tree Map pentru modelarea polinoamelor, am decis ca valoarea să fie coeficientul, iar cheia să fie gradul. Tree Map este o alegere avantajoasă din cauza complexității operațiilor de prelucrare a acestui tip de date și din cauză că datele sunt păstrate ordonat.

4. Implementare

Clasa Polynomial

Câmpuri :

- TreeMap<Integer, Double> polynomial.

Constructori :

- Polynomial().

Metode accesoare:

- getPolynomial().

Metode :

- toString() : returnează polinomul sub forma unui șir de caractere ;
- addTerm(int degree, double coef) : Adaugă un nou termen la polinom. Dacă gradul există deja în polinom, se adaugă coeficientul la cel existent.

```
public void addTerm(int degree, double coef) {
    if(degree>=0) {
        if (polynomial.containsKey(degree)) {
            polynomial.put(degree, polynomial.get(degree) + coef);
        } else {
            polynomial.put(degree, coef);
        }
    }
}
```

Clasa Operation

Constructori :

- Operation().

Metode :

- add(Polynomial, Polynomial): adună două polinoame adăugând toți termenii acestora la un rezultat, folosind metoda addTerm;
- sub(Polynomial, Polynomial): scade două polinoame;

```
public static Polynomial sub(Polynomial pol1, Polynomial pol2) {
    Polynomial res = new Polynomial();
    int degree;
    double coef;
    for (Map.Entry<Integer, Double> p : pol1.getPolynomial().entrySet()) {
        degree = p.getKey();
        coef = p.getValue();
        res.addTerm(degree, coef);
    }

    for (Map.Entry<Integer, Double> p : pol2.getPolynomial().entrySet()) {
        degree = p.getKey();
        coef = p.getValue();
        res.addTerm(degree, -coef);
    }
    return res;
}
```

- multiply(Polynomial, Polynomial): înmulțește două polinoame. Această metodă parcurge fiecare termen din primul polinom și îl înmulțește cu fiecare termen din al doilea polinom;
- divide(Polynomial, Polynomial): împarte două polinoame, dacă se încearcă împărțirea cu 0 se va afișa un mesaj de eroare;
- deriv(Polynomial): calculează derivata unui polinom și returnează rezultatul. Această metodă parcurge fiecare termen din polinom și calculează derivata termenului respectiv;
- integrat(Polynomial): calculează integrata unui polinom și returnează rezultatul. Această metodă parcurge fiecare termen din polinom și calculează integrata; termenului respectiv.
- verify(String): verifică dacă datele introduse de utilizator sunt valide;
- transform(String): transformă datele introduse de utilizator în polinoame.

Clasa View

Câmpuri :

- JTextField pol1TxtField: zona de text unde se introduce primul polinom ;
- JTextField pol2TxtField: zona de text unde se introduce al doilea polinom ;
- Controller controller: pe baza modelului MVC ;

Constructori :

- View(): apelează metoda de inițializare a ferestrei pe care o să o vadă utilizatorul.

Metode :

- Initialize() : apelează toate metodele care se ocupă de instrumentele de pe interfața grafică și de designul acestora ;
- helpView() : adaugă și personalizează butonul „Help”, trimite un semnal spre Controller când butonul este apăsat;
- alte metode similare cu helpView().

```
private void helpView(){
    helpPanel = new JPanel();
    helpPanel.setBounds( x: 368,  y: 13,  width: 64,  height: 38);
    resPanel.add(helpPanel);
    helpPanel.setLayout(null);
    helpPanel.setBorder(new LineBorder(new Color( r: 40,  g: 149,  b: 44),  thickness: 4,  roundedCorners: true));
    helpPanel.setBackground(new Color( r: 128,  g: 255,  b: 128));

    JLabel helpLbl = new JLabel( text: "Help");
    helpLbl.setHorizontalAlignment(SwingConstants.CENTER);
    helpLbl.setFont(new Font( name: "Cambria", Font.BOLD,  size: 17));
    helpLbl.setBounds( x: 10,  y: 10,  width: 44,  height: 18);
    helpPanel.add(helpLbl);

    helpPanel.addMouseListener((MouseAdapter) mousePressed(e) -> {
            controller.helpPanelPressed();
    });
}
```

Metode accesoare:

- getPol1TxtField(): oferă acces la textul introdus de utilizator în zona alocată primului polinom;
- getPol2TxtField(): oferă acces la textul introdus de utilizator în zona alocată celui de al doilea polinom;

```
public String getPol2Txt() {
    return pol2TxtField.getText();
}
```

- getResTxtField(): oferă acces la zona alocată rezultatului.

Clasa Controller

Câmpuri :

- View view : MVC.

Constructori :

- Controller(View).

Metode:

- Metodele primesc semnal din view când utiliztorul interacționează cu obiectele și decid cum se schimbă partea pe care acesta o vede.
- helpPanelPressed() : stabilește cum se modifică interfața grafică la apăsarea butonului.

```
public void helpPanelPressed() {
    view.getResTxtField().setText("Data form, without spaces, use just x: x-5+x^8");
}
```

Clasa App (Main)

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

Clasa AppTest

În această clasă se testează dacă operațiile au fost implementate corect.

Câmpuri :

- Polynomial p1 ;
- Polynomial p2.

Metode :

- setUpBefore(): această metodă inițializează polinoamele înainte de inceperea executării testelor ;
- testTransform(): testează dacă datele sunt transformate corect din String în polinoame ;
- testAdd(): testează adunarea ;
- alte metode similare.

5. Rezultate

Implementarea operatiilor cu polinoame este verificată prin teste scrise pentru fiecare operatie din clasa Operation folosind framework-ul JUnit. Înainte de începerea testelor am inițializat două polinoame.

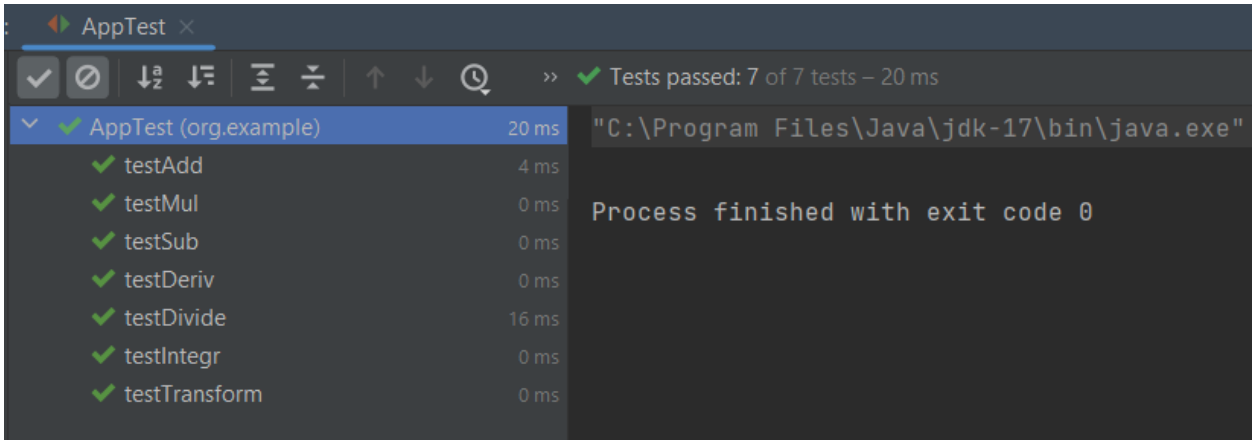
```
@BeforeClass
public static void setUpBefore() {
    p1 = new Polynomial();
    p1.addTerm( degree: 3,  coef: 2.0);
    p1.addTerm( degree: 2,  coef: 3.0);
    p1.addTerm( degree: 1,  coef: -4.0);
    p1.addTerm( degree: 0,  coef: 1.0);
    p1.addTerm( degree: 1,  coef: 3.0);
    p1.addTerm( degree: 9,  coef: 0.0);

    p2 = new Polynomial();
    p2.addTerm( degree: 3,  coef: 7.0);
    p2.addTerm( degree: 3, coef: -2.0);
    p2.addTerm( degree: 2,  coef: 3.0);
    p2.addTerm( degree: 5,  coef: -4.0);
    p2.addTerm( degree: 0,  coef: 8.0);
    p2.addTerm( degree: 9,  coef: 3.0);
}
```

În metodele testelor am verificat dacă rezultatul operației este egal cu cel așteptat.

```
@Test
public void testAdd() {
    Polynomial p3= Operation.add(p1,p2);
    Polynomial res=new Polynomial();
    res.addTerm( degree: 0,  coef: 9.0);
    res.addTerm( degree: 1,  coef: -1.0);
    res.addTerm( degree: 2,  coef: 6.0);
    res.addTerm( degree: 3,  coef: 7.0);
    res.addTerm( degree: 5,  coef: -4.0);
    res.addTerm( degree: 9,  coef: 3.0);
    assertEquals((p3.getPolynomial()).equals(res.getPolynomial()), actual: true);
}
```

```java
@Test
public void testTransform(){
    Polynomial p=Operation.transform( exp: "6y^2");
    int res;
    if(p.getPolynomial().containsKey(-1)==true ) {
        res=0;
    }
    else {
        res = 1;
    }
    assertEquals(res, actual: 0);
}
```

```java
@Test
public void testDivide(){
    Polynomial p1=Operation.transform( exp: "6x^2");
    Polynomial p2=Operation.transform( exp: "0");
    ArrayList<Polynomial>res=Operation.divide(p1,p2);
    assertEquals(res.get(0).getPolynomial().containsKey(-1), actual: true);
}
```



Toate testele s-au încheiat cu succes!

6. Concluzii

Din punctul meu de vedere, acest proiect te ajută să îmbini utilul cu plăcutul deoarece rezultatul acestuia este un calculator funcțional prin realizarea căruia am învațat lucruri noi. Am reușit să înțeleg mai bine conceptul de polinoame și am învățat să lucrez cu Tree Map. Această temă mi-a dezvoltat abilitățile de rezolvare a problemelor și de gândire algoritmică. Am realizat că doar testând o aplicație poți să descoperi eventuale excepții de care nu ai ținut cont initial.

Posibile dezvoltari ulterioare:

- Evaluarea polinoamelor intr-un anumit punct;
- Extinderea implementarii pentru a permite lucrul cu polinoame cu coeficienți rationali și grade întregi;
- Îmbunătățirea interfeței grafice.

7. Bibliografie

1) *https://dsrl.eu/courses/pt/*
2) *https://www.javatpoint.com/java-treemap*
3) *https://regex101.com/r/nI5oA5/6*
4) *https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html*