



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

DOCUMENTAȚIE

TEMA 1

Calculator de polinoame

Boloș Andrei Nicolae
GRUPA: 30236

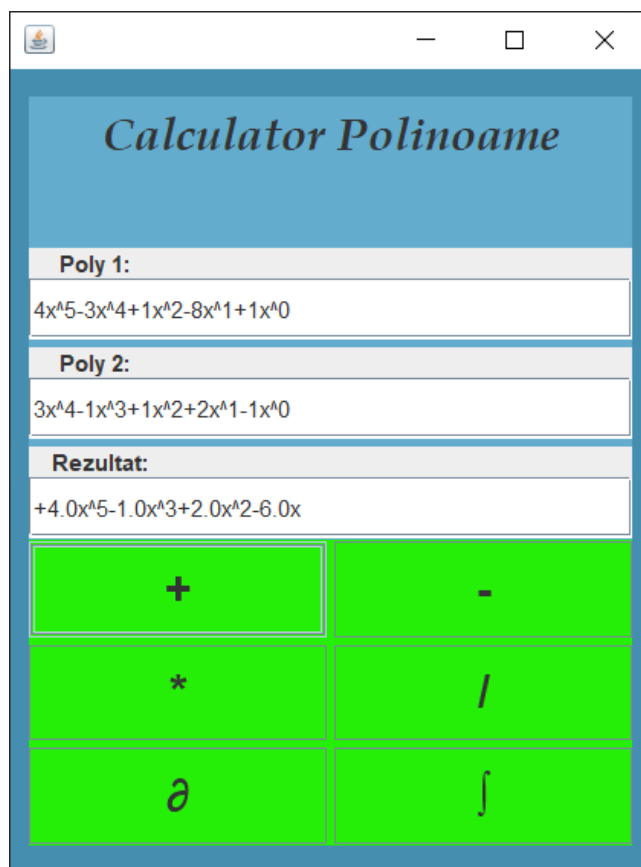
CUPRINS

CUPRINS.....	2
1. Obiectivul temei.....	3
2. Analiza problemei, modelare.....	4
3. Proiectare.....	5
4. Implementare	7
5. Testare.....	10
6. Concluzii.....	10
7. Bibliografie	12

1. Obiectivul temei

Obiectivul principal al temei reprezintă proiectarea și implementarea unui calculator de polinoame într-o aplicație Java.

Utilizatorul va interacționa cu acest calculator folosind o interfață grafică interactivă:



Calculator Polinoame

Poly 1:
 $4x^5-3x^4+1x^2-8x^1+1x^0$

Poly 2:
 $3x^4-1x^3+1x^2+2x^1-1x^0$

Rezultat:
 $+4.0x^5-1.0x^3+2.0x^2-6.0x$

Buttons:
+, -, *, /, ∂, ∫

2. Analiza problemei, modelare

La prima vedere, se observă că este necesară o interfață grafică ce permite interacțiunea dintre utilizator și logica aplicației ascunsă de programator. Aceasta ar trebui să permit introducerea de la tastatură a celor două polinoame, un spațiu destinat afișării rezultatului și niște butoane ce traduc intențiile utilizatorului în comenzi de executat de către calculator.

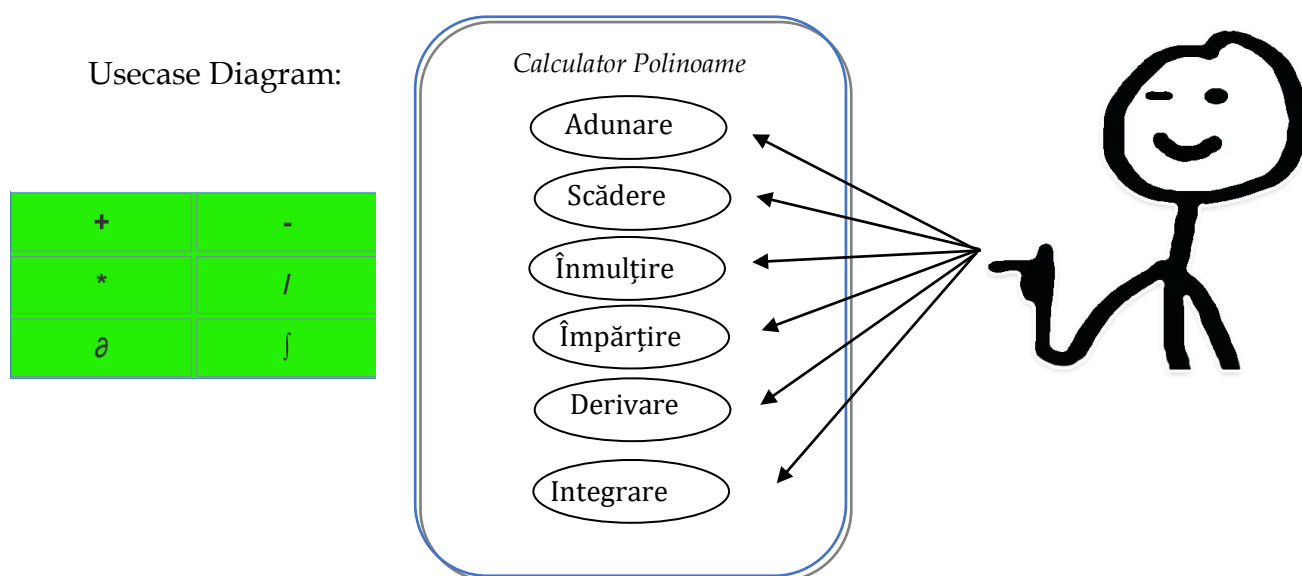
Calculatorul are disponibile următoarele operații:

- ➔ Adunarea a două polinoame;
- ➔ Scăderea a două polinoame;
- ➔ Înmulțirea a două polinoame;
- ➔ Împărțirea a două polinoame;
- ➔ Derivarea unui polinom;
- ➔ Integrarea unui polinom;

Un Polinom este format din mai multe Monoame, și este de forma:

$$a_n * x^n + a_{n-1} * x^{n-1} + \dots + a_1 * x + a_0$$

Utilizatorul introduce un polinom valid, de forma $\sum a_i X^{n_i}$, unde a_i reprezintă coeficientul, respectiv n_i reprezintă exponentul (ex. $1x^2 - 2x^1 + 5x^0$ ce înseamnă $x^2 + 2x + 5$) și la apăsarea pe butonul ce reprezintă operația dorită se va genera un răspuns și va fi afișat într-o căsuță text.



unde:

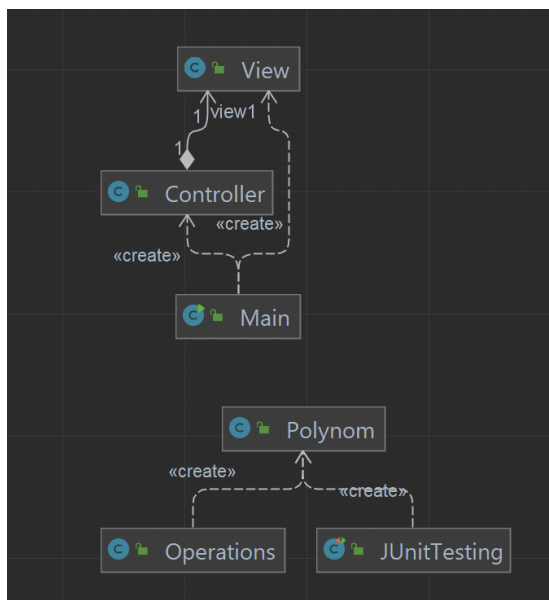
- + Adunare – este operația de adunare a două polinoame;
- Scădere – este operația de scădere a două polinoame;
- * Înmulțire - este operația de înmulțire a două polinoame;
- / Împărțire - este operația de împărțire a două polinoame;
- ∂ Derivare - este operația de derivare a unui polinom;
- \int Integrare - este operația de integrare a unui polinom.

3. Proiectare

Diagrama UML (Unified Modeling Language) reprezintă structura logică a proiectului. Specifică împărțirea în pachete, clase și relații.

Ea poate conține și atributele, constructorii și metodele claselor.

Diagrama UML:



În clasa **Main** se instanțiază un obiect din clasa **View**, și pe acest obiect se apelează constructorul clasei **Controller**. Așa se urmează modelul **MVC** al programării.

Un Polinom este salvat în memorie sub forma unui **HashMap** a cărui *cheie* o reprezintă *exponentul* (un număr întreg, n), iar *valoarea* de la adresa respective o constituie *coeficientul* (un număr rational, a_n). Fiecare *entry* este de forma $a_n * x^n$.

Clasa *Monom*, care ar avea câmpurile *coeficient* și *exponent* este opțională deoarece un **HashMap** este suficient, având disponibilă varianta de a reține *exponentul* în câmpul *key* și *coeficientul* în câmpul *value*.

Așadar, modelul de lucru al aplicației se rezumă la clasa **Polynom** ce conține ca atribut un *HashMap*.

Atât operațiile cât și clasa de *testare* se bazează pe clasa **Polynom**, făcând instanțe de polinoame.

Proiectul va conține 3 Pachete pentru implementare:

- *model*;
- *controller*;
- *view*;

Pachetul *model* conține clasele:

- *Main*;
- *Polynom*;
- *Operations*;

Pachetul *controller* conține clasele:

- *Controller*;

Pachetul *view* conține clasele:

- *View*;

4. Implementare

Pachetul *model*:

Clase:

1. Main

Clasa *Main* conține inițializarea unei interfețe grafice și inițializarea unui controller care are rol de management pentru operațiile pe care le dezvoltă calculatorul.

```
public class Main {
    public static void main(String[] args) {
        View view1 = new View();
        view1.setVisible(true);
        Controller controller1 = new Controller(view1);
    }
}
```

2. Polynom

Clasa *Polynom* are ca atribut un *HashMap* a cărei cheie este exponentul (un număr întreg, n), iar valoarea de la adresa respective o constituie coeficientul (un număr rațional, a_n). Fiecare *entry* este de forma $a_n * x^n$, și reprezintă un *Monom*.

```
private HashMap<Integer, Double>poly;
```

Conține un constructor fără parametri și unul care primește ca parametru un *String*, pe care îl parsează, separând coeficienții și exponenții. Se folosesc Expresii Regulate (*regex*). Acestea despart textul în mai multe părți ce constituie caracterele definitorii ale unui monom: $X^$, și eventuale semne (+,-) și numere ce reprezintă coeficienții și exponenții.

Se face *split* la întâlnirea caracterului ^, iar ce e înainte de acesta se parsează într-un *Integer*, cât este valid, iar ce e după acel caracter se introduce la coeficient ca un număr rațional.

În gruparea "\\^", primul '\\' face *escape* celui de-al doilea '^', iar '^' interpretat face *escape* la '^'

```
Pattern pattern = Pattern.compile("([+-]?[^-+]+)");
Matcher matcher = pattern.matcher(expr);
HashMap<Integer, Double>poly = new HashMap<Integer, Double>();
while (matcher.find()) {
    String[] tokens = matcher.group().split("\\^");
    double coef =
    Double.parseDouble(tokens[0].substring(0, tokens[0].length()-1));
```

```
int exp = Integer.parseInt(tokens[1]);
poly.put(exp, coef);
```

În această clasă este suprascrisă funcția *toString()* pentru a afișa un polinom pe ecran așa cum este dorit.

3. Operations

Clasa *Operations* implementează logica operațiilor care alcătuiesc calculatorul.

Pentru *adunarea* a două polinoame se creează un nou *polinom* care va fi returnat, inițial cu valoarea polinomului *a*. Se iau două instanțe ale polinoamelor primite ca *TreeMap* și se copiază valorile din polinomul *b* în rezultat pentru a se marca pozițiile exponenților lui *b* disponibile pentru comparare cu exponenții lui *a*. Pentru fiecare monom din polinomul *b* care are același coeficient cu cel din *a*, atunci monomul rezultat va avea coeficientul format din suma celor 2 coeficienți. Dacă în urma adunării, un monom are coeficient nul, atunci el se scoate din *HashMap*-ul de monoame a polinomului, urmând a nu mai fi afișat.

```
public static Polynom suma(Polynom a, Polynom b){
    Polynom rez= new Polynom(a.getPoly());
    TreeMap<Integer,Double> a1 = new TreeMap<Integer,Double>(a.getPoly());
    TreeMap<Integer,Double> b1 = new TreeMap<Integer,Double>(b.getPoly());
    int key = -1;
    for (Map.Entry<Integer, Double> entry2: b1.entrySet()) {
        rez.getPoly().put(entry2.getKey(), entry2.getValue());
    }
    for (Map.Entry<Integer, Double> entry2: b1.entrySet()) {
        for (Map.Entry<Integer, Double> entry1: a1.entrySet()) {
            if (entry2.getKey().equals(entry1.getKey())) {
                rez.getPoly().put(entry2.getKey(), entry1.getValue() + entry2.getValue());
            }
        }
        for (Map.Entry<Integer, Double> ello: rez.getPoly().entrySet()) {
            if(ello.getValue() == 0)
                key = ello.getKey();
        }
        rez.getPoly().remove(key);
    }
    // System.out.println(rez.toString());
    return rez;
}
```

Asemănător se procedează și pentru *scăderea* a două polinoame. Singurele diferențe față de adunare sunt că din *TreeMap*-ul *b1* se copiază coeficienții cu semn schimbat, iar când se adaugă la rezultatul final, vlaorile coeficienților celui de-al doilea polinom se scad în loc să se adune.

La *înmulțire*, fiecare monom al primului polinom se înmulțește cu fiecare monom al celui alt polinom. La înmulțirea a două monoame,

monomul rezultat va avea *exponentul* format din *suma* celor doi exponenți ai factorilor, iar *coeficientul* va fi format din produsul celor doi coeficienți ai monoamelor participante la înmulțire. Dacă în HashMap este populată deja celula de la adresa exponentului proaspăt calculate, atunci valoarea coeficientului se înlocuiește cu suma dintre vechea valoare și noua valoare.

```
for (Map.Entry<Integer, Double> entry2: b1.entrySet()) {
    for (Map.Entry<Integer, Double> entry1: a1.entrySet()) {
        int newKey = entry1.getKey()+entry2.getKey();
        if (rez.getPoly().containsKey(newKey)) {
            rez.getPoly().replace(newKey, rez.getPoly().get(entry1.getKey() +
entry2.getKey()) + entry1.getValue()*entry2.getValue());
        }
        else {
            rez.getPoly().put(entry1.getKey() + entry2.getKey(), entry1.getValue()
* entry2.getValue());
        }
    }
}
```

Derivarea unui polinom se face după formula:

$$(x^n)' = n \cdot x^{n-1};$$

```
public static Polynom derivarea(Polynom a) {
    Polynom rez = new Polynom();
    TreeMap<Integer, Double> a1 = new TreeMap<Integer, Double>(a.getPoly());
    for (Map.Entry<Integer, Double> entry: a1.entrySet()) {
        if (entry.getKey() >= 2) {
            rez.getPoly().put(entry.getKey()-1, entry.getValue() * entry.getKey());
        }
        else {
            rez.getPoly().put(0, entry.getValue());
        }
    }
    return rez;
}
```

Integrarea unui polinom se face după formula:

$$\int a \cdot x^n = a \cdot x^{n+1} / (n+1);$$

Afișarea rezultatului se mai putea îmbunătăți: să nu se afișeze '+' înainte de primul monom (din suprascrierea metodei *toString()*).

Pachetul *controller*:

Clasa: **Controller**

Clasa *Controller* conține clase ce implementează acțiunile care au loc la apăsarea butoanelor.

```
static class sumaListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Se efectueaza +");
        String a = view1.getField1().getText();
        //System.out.println(a);
        Polynom poly1 = new Polynom(a);
```

```

String b = view1.getField2().getText();
Polynom poly2 = new Polynom(b);
view1.getField3().setText(Operations.suma(poly1, poly2).toString());
//      System.out.println(Operations.suma(poly1, poly2).toString());
    }
}

```

Pachetul *view*:

Clasa: *View*

Clasa *View* are rolul de a implementa aspectul visual al interfeței cu care interacționează utilizatorul. Aceasta este formată dintr-un *Frame* care conține mai multe *Panel*-uri, ce sunt formate la rândul lor din elemente precum *TextField*, *Label* și *Butoane*. Toate aceste elemente sunt editate astfel încât să aibă un anumit aspect.

5. Testare

Partea de testare reprezintă verificarea funcționalității calculatorului după ce acesta a fost implementat. Se rulează cu ajutorul bibliotecii JUnit câte un test pentru fiecare operație. Dacă rezultatul așteptat coincide cu cel generat de calculator, atunci testul este unul de success, altfel eșuează.

```

public class JUnitTesting {
    @Test
    public void adunareTest(){
        Polynom p1=new Polynom("4x^5-3x^4+1x^2-8x^1+1x^0");
        Polynom p2=new Polynom("3x^4-1x^3+1x^2+2x^1-1x^0");
        String rezultat = new String("+4.0x^5-1.0x^3+2.0x^2-6.0x");
        assertEquals(rezultat, Operations.suma(p1,p2).toString());
    }
}

```

✓ JUnitTesting	33 ms
✓ integrateTest()	29 ms
✓ scadereTest()	2 ms
✓ derivareTest()	1 ms
✓ adunareTest()	
✓ inmultireTest()	1 ms

6. Concluzii

Acest proiect este o implementare a unui calculator de polinoame. Poate fi util în domeniul matematicii, al fizicii sau al informaticii. Această implementare poate fi îmbunătățită atât visual cât și din punct de vedere al performanței.

Se pot arunca și intercepta excepții ale utilizării, precum introducerea unui text invalid în câmpul pentru citirea unui polinom. Astfel se pot afișa niște texte de îndrumare către utilizator încât să introducă polinoamele în forma lor acceptată spre procesare ulterioară.

7. Bibliografie

Link Git: https://github.com/PT202330227BolosAndrei/PT202330227BolosAndrei-PT2023_30227_Bolos_Andrei_Assignment_1

Link Resurse:

<https://stackoverflow.com/questions/9752665/extracting-polynomial-variables-from-string-input-via-a-constructor>

<https://dsrl.eu/courses/pt/>

https://dsrl.eu/courses/pt/materials/A1_Support_Presentation.pdf

<https://dsrl.eu/mantal/pt/>

<http://tutorials.jenkov.com/java-regex/matcher.html>

<https://docs.oracle.com/javase/tutorial/essential/regex/groups.html>

<https://en.wikipedia.org/wiki/Polynomial>