DOCUMENTAȚIE

Tema 1

Lihaciu Oana

Grupa 30225

CUPRINS

1. Obiectivul temei

Obiectivul principal:

În cadrul proiectului propunem dezvoltarea unui calculator de polinoame (cu interfață) cu ajutorul căruia utilizatorii vor putea vizualiza răspunsul introducând două polinoame și selectând operația dorită.

Obiectivele secundare:

Mai jos sunt prezentați pașii necesari pentru atingerea obiectivului principal, indicând în care capitole ale documentației urmează a fi dezvoltate:

- Analizarea problemei și identificarea cerințelor (Capitolul 2)
- Proiectarea calculatorului polinomial (Capitolul 3)
- Implementarea calculatorului polinomial (Capitolul 4)
- Testarea (Capitolul 5)

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

- **Problema**

  - principala problemă o reprezintă dificultatea cu care se realizează calcularea polinoamelor. Sunt calcule dese, lungi și care consumă foarte mult timp

- **Soluția**

  - soluția reprezintă un calculator care poate îndeplini următoarele cerințe:

  a) Cerințe funcționale:

  - Calculatorul polinomial trebuie să permită utilizatorilor să introducă d a polinoame

  lculatorul polinomial trebuie să permită utilizatorilor să selecteze o operaț e matematica

  - C latorul polinomial trebuie să permită utilizatorilor să vizualizeze rezultatu operațiilor

  - Calc orul polinomial trebuie să fie capabil să efectueze o serie de operații mat matice: adunarea, scăderea, înmulțirea, împărțirea, integrarea ș derivarea polinoamelor.

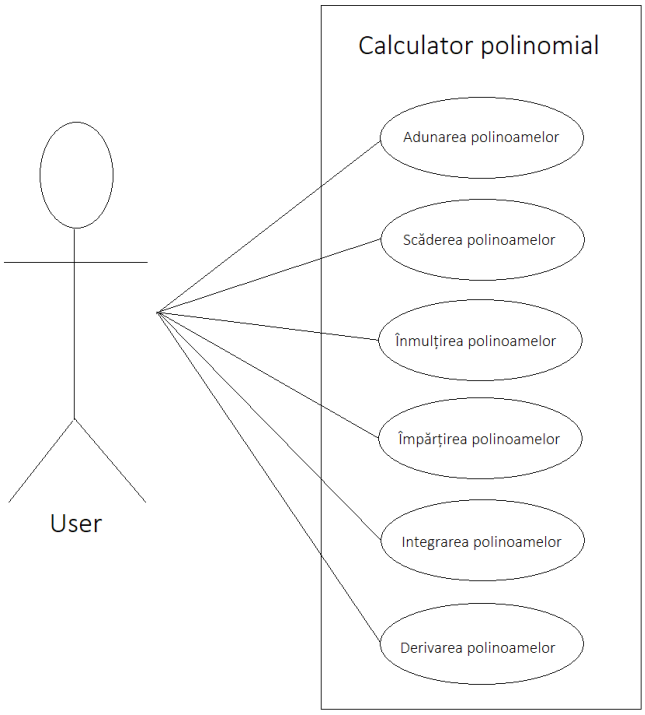
  - Calcula l polinomial trebuie să returneze rezultate cu precizie de o zecimală, în ca ul coeficienților rționali

  b) Cerințe cționale

  - Calculatorul pol ial trebuie să fie intuitiv și ușor de folosit de către utilizatori

  - alculatorul polino trebuie să fie capabil să lucreze cu numere mari fără un impact semnifi ativ asupra timpului de răspuns

- **Cazuri de utilizare**



1. **Caz de utilizare:** Adăugare polinoame
   Actor principal: Utilizator
   Scenariu principal de succes:

- Utilizatorul introduce cele două polinoame în interfața grafică.
- Utilizatorul selectează operația "+".
- Calculatorul polinomial efectuează adunarea celor două polinoame și afișează rezultatul.

2. **Caz de utilizare:** Scădere polinoame
   Actor principal: Utilizator
   Scenariu principal de succes:

- Utilizatorul introduce cele două polinoame în interfața grafică.
- Utilizatorul selectează operația "-".
- Calculatorul polinomial efectuează scăderea celor două polinoame și afișează rezultatul.

3. **Caz de utilizare:** Înmulțire polinoame
   Actor principal: Utilizator
   Scenariu principal de succes:

- Utilizatorul introduce cele două polinoame în interfața grafică.
- Utilizatorul selectează operația "*".
- Calculatorul polinomial efectuează înmulțirea celor două polinoame și afișează rezultatul.

4. **Caz de utilizare:** Împărțire polinoame
   Actor principal: Utilizator
   Scenariu principal de succes:

- Utilizatorul introduce cele două polinoame în interfața grafică.
- Utilizatorul selectează operația "/".
- Calculatorul polinomial efectuează împărțirea celor două polinoame și afișează câtul și restul.

5. **Caz de utilizare:** Integrare polinom
   Actor principal: Utilizator
   Scenariu principal de succes:

- Utilizatorul introduce cele polinomul în interfața grafică.
- Utilizatorul selectează operația "Integrare".
- Calculatorul polinomial efectuează integrarea polinomului.

Secvență alternativă: Polinoame incorecte

- Utilizatorul introduce polinomul cu rol de împărțitor ca fiind “0”
- Câtul devine “err”

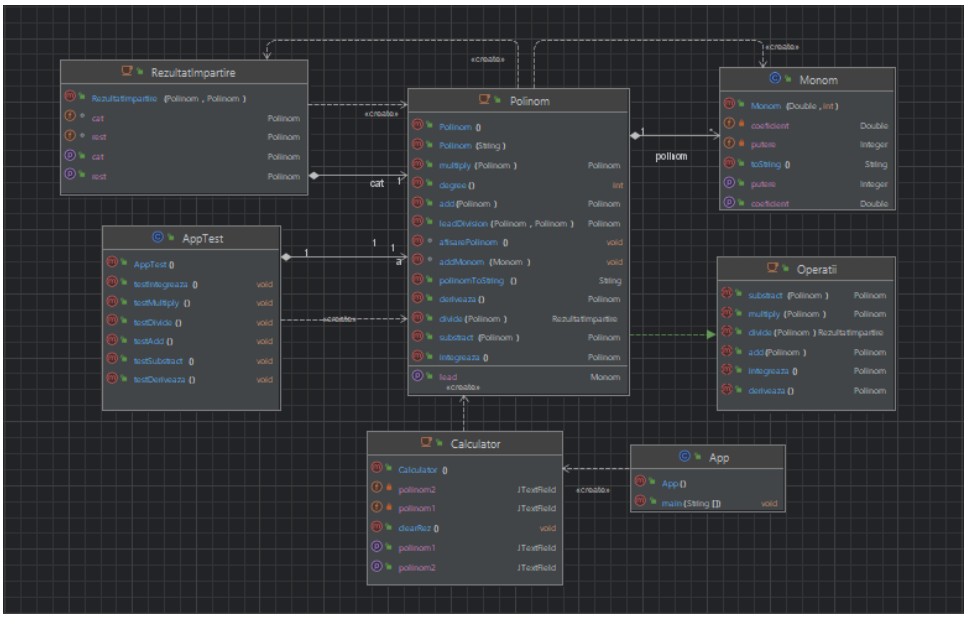
6. **Caz de utilizare:** Derivare polinom
   Actor principal: Utilizator
   Scenariu principal de succes:

- Utilizatorul introduce cele polinomul în interfața grafică.
- Utilizatorul selectează operația "Derivare".
- Calculatorul polinomial efectuează derivarea polinomului.

3. Proiectare

În faza de proiectare a aplicației noastre de calculator polinomial, am adoptat o abordare orientată pe obiecte (OOP), care ne-a permis să structurăm logic și eficient întregul sistem.
Am elaborat diagramele UML de clase și de pachete pentru a vizualiza relațiile dintre entități și pentru a organiza codul într-o manieră coerentă și ușor de înțeles.

- **Diagrama UML**



- **Structurile de date**

- HashMap: Folosită în clasa **Polinom** pentru a stoca monomii asociindu-le puterile.
- String: Folosit în diferitele clase pentru a reprezenta expresiile polinomiale și pentru a realiza operații de parsare și manipulare a acestora.

- **Interfețe**

  Am folosit o singură interfață: “Operatii”, implementată de clasa “Polinom”

```java
package polinoame;

public interface Operatii {
    public Polinom add(Polinom b);
    public Polinom substract(Polinom b);
    public Polinom multiply(Polinom b);
    public RezultatImpartire divide(Polinom b);
    public Polinom integreaza();
    public Polinom deriveaza();
```

4. Implementare

Clasa Monom:

- **Câmpuri:**
  coeficient: Double - coeficientul monomului.
  putere: Integer - puterea monomului.

- **Metode:**
  Monom(Double coeficient, int putere): Constructor pentru inițializarea unui monom cu coeficientul și puterea date.
  getCoeficient() : Double: Returnează coeficientul monomului.
  setCoeficient(Double coeficient) : void: Setează coeficientul monomului.
  getPutere() : Integer: Returnează puterea monomului.
  setPutere(Integer putere) : void: Setează puterea monomului.
  toString() : String: Returnează o reprezentare sub formă de șir de caractere a monomului.

Clasa Polinom:

- **Campuri:**
  polinom: HashMap<Integer, Monom> - reprezintă polinomul, cu puterile monomilor ca chei și monomii ca valori.

- **Metode:**
  Polinom(): Constructor pentru inițializarea unui polinom gol.
  Polinom(String exp): Constructor pentru inițializarea unui polinom pe baza unui șir de caractere care reprezintă expresia polinomială.
  addMonom(Monom a): Adaugă un monom în polinom.
  add(Polinom b) : Polinom: Adaugă doi polinoame.
  substract(Polinom b) : Polinom: Scade două polinoame.
  multiply(Polinom b) : Polinom: Înmulțește două polinoame.
  divide(Polinom b) : RezultatImpartire: Împarte două polinoame și returnează rezultatul sub formă de obiect **RezultatImpartire**.
  integreaza() : Polinom: Efectuează integrarea polinomului.
  deriveaza() : Polinom: Efectuează derivarea polinomului.
  afisarePolinom(): Afisează polinomul la consolă.
  polinomToString() : String: Returnează o reprezentare sub formă de șir de caractere a polinomului.

degree() : int: Returnează gradul polinomului.
getLead() : Monom: Returnează monomul cu cea mai mare putere.
leadDivision(Polinom dp, Polinom ip) : Polinom: Efectuează împărțirea polinoamelor prin conducători.

Clasa RezultatImpartire:

- **Campuri:**
  cat: Polinom - reprezintă câtul împărțirii.
  rest: Polinom - reprezintă restul împărțirii.

- **Metode:**
  RezultatImpartire(Polinom cat, Polinom rest): Constructor pentru inițializarea unui obiect **RezultatImpartire**.
  getCat() : Polinom: Returnează câtul împărțirii.
  getRest() : Polinom: Returnează restul împărțirii.

Interfața Operatii:

- **Metode:**
  add(Polinom b) : Polinom: Adaugă doi polinoame.
  substract(Polinom b) : Polinom: Scade două polinoame.
  multiply(Polinom b) : Polinom: Înmulțește două polinoame.
  divide(Polinom b) : RezultatImpartire: Împarte două polinoame și returnează rezultatul sub formă de obiect **RezultatImpartire**.
  integreaza() : Polinom: Efectuează integrarea polinomului.
  deriveaza() : Polinom: Efectuează derivarea polinomului.

Clasa Calculator (Interfața Utilizatorului):

- **Campuri:**
  Diverse componente GUI: JTextField, JButton, JLabel - reprezintă câmpurile de text, butoanele și etichetele interfeței grafice.
- **Metode:**
  Metodele care asociază acțiuni butoanelor din interfață pentru efectuarea operațiilor pe polinoame, cum ar fi **addActionListener**, **clearRez**, etc.

5. Rezultate

Pentru a evalua corectitudinea funcționării operațiilor implementate asupra polinoamelor, am efectuat o serie de teste unitare folosind framework-ul JUnit. Am utilizat aceleași polinoame în toate testele, și anume "2x^4 + 2x^3" și "-4x + 4". Scenariile de testare au inclus:

- **Adunare:** Testarea operației de adunare a două polinoame pentru a verifica dacă rezultatul obținut este cel așteptat.
- **Scădere:** Verificarea corectitudinii operației de scădere a două polinoame și confirmarea rezultatului în conformitate cu așteptările.
- **Înmulțire:** Testarea operației de înmulțire a două polinoame pentru a asigura funcționarea corectă a implementării.
- **Împărțire:** Verificarea corectitudinii operației de împărțire a două polinoame și confirmarea rezultatului corespunzător. Pentru acest test, am analizat și rezultatul câtului și al restului împărțirii.
- **Integrare:** Evaluarea funcționării corecte a operației de integrare a unui polinom.
- **Derivare:** Testarea operației de derivare a unui polinom pentru a confirma obținerea rezultatului așteptat.

Aceste teste au acoperit diverse aspecte ale operațiilor matematice aplicate asupra polinoamelor și au asigurat corectitudinea funcționării acestora în cadrul aplicației.

```java
package test;

import org.junit.Test;
import polinoame.Polinom;
import polinoame.RezultatImpartire;

import static org.junit.Assert.*;
public class AppTest{

    Polinom a=new Polinom( exp: "2x^4+2x^3");
    Polinom b=new Polinom( exp: "-4x+4");

    @Test
    public void testAdd() {
        Polinom rez=a.add(b);
        assertEquals( expected: "+2x^4+2x^3-4x+4", rez.polinomToString());
    }
```

6. Concluzii

Acest proiect m-a ajutat să înțeleg și să aplic unele concepte și tehnologii importante în programare. Am învățat cum să utilizez expresiile regulate (regex) pentru analiza și parsarea textului, precum și modul de utilizare a structurii de date HashMap pentru stocarea și manipularea datelor. De asemenea, am câștigat experiență în testarea unitară cu ajutorul framework-ului JUnit, ceea ce m-a ajutat să verific și să validez corectitudinea funcționării codului în mod automat.

Pentru o viitoare îmbunătățire a calculatorului, se pot lua în considerare următoarele idei:

- Extinderea funcționalității pentru a permite utilizatorilor să lucreze cu puteri reale, numere complexe și fracții.
- Implementarea unui mecanism de gestionare a erorilor mai robust, care să ofere feedback clar și concis utilizatorilor în cazul introducerii unor date invalide.
- Adăugarea unei interfețe grafice mai interactive și intuitive, care să ofere o experiență mai plăcută și mai ușor de utilizat.
- Optimizarea algoritmilor existenți pentru a îmbunătăți performanța și eficiența calculatorului.
- Extinderea documentației și adăugarea de exemple practice pentru a ajuta utilizatorii să înțeleagă mai bine modul de utilizare a calculatorului și a funcționalităților acestuia.

Aceste îmbunătățiri ar putea să ofere utilizatorilor o experiență mai bună în utilizarea calculatorului de polinoame, acoperind o gamă mai largă de scenarii și necesități matematice.

BIBLIOGRAFIE

https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html

https://www.geeksforgeeks.org/write-regular-expressions/

https://www.w3schools.com/java/java_hashmap.asp