DOCUMENTATIE

TEMA *1*

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

NUME STUDENT: Toader Eric-Stefan
GRUPA: 302210

CUPRINS

1. Obiectivul temei

Obiectivul principal al temei este implementarea folosind limbajul de programare Java a unui calculator de polinoame cu o interfata grafica dedicata, in care utilizatorul poate introduce polinoame si selecta diferite operatii matematice, precum adunarea, scaderea, inmultirea, etc.

Obiectivele secundare ale temei sunt:

1. Definirea claselor Polinom si Monom: se va crea clasa Monom ce contine informatiile unui monom (coeficient de tip numar real si gradul de tip numar intreg), cu metodele necesare si o metoda toString ce va asigura o listare prietenoasa a monoamelor unui polinom; se va crea clasa Polinom, ce contine o lista de Monoame
2. Implementarea operatiilor de adunare si scadere: dezvoltarea unui algoritm ce ia ca input 2 polinoame si va returna polinomul rezultat din adunarea, respectiv scaderea celor doua polinoame de intrare; aceste operatii se vor defini ca metode statice in clasa Operatii
3. Implementarea operatiei de inmultire: dezvoltarea unui algoritm ce ia ca input 2 polinoame si va returna polinomul rezultat din inmultirea celor doua polinoame de intrare; aceasta operatie se va defini ca metoda statica in clasa Operatii
4. Implementarea operatiei de impartire: dezvoltarea unui algoritm ce ia ca input 2 polinoame si va returna un obiect de tipul clasei RezultatImpartire ce contine doua polinoame (catul, respectiv restul impartirii) ; aceasta operatie se va defini ca metoda statica in clasa Operatii
5. Implementarea operatiilor de integrare si derivare: dezvoltarea unui algoritm ce ia ca input un singur polinom si va returna polinomul rezultat din integrarea, respectiv derivarea polinomului de intrare; aceste operatii se vor defini ca metode statice in clasa Operatii
6. Parsarea stringurilor la un obiect de tip Polinom: dezvoltarea unui algoritm ce ia ca input un sir de caractere si va returna un Polinom rezultat din prelucrarea datelor de intrare; aceasta metoda se va defini ca metoda statica in clasa Polinom
7. Realizarea unei interfete grafice prietenoase: dezvoltarea unei interfete grafice intuitive care sa faca posibila introducerea de polinoame si calcularea tuturor operatiilor matematice, in conditiile in care datele introduse de utilizator sunt identificate ca fiind polinoame valide

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

Pentru a rezolva cerintele problemei, va trebui sa realizam o interfata grafica prietenoasa si intuitiva ca utilizatorul sa poata introduce doua polinoame si sa obtina rezultatul oricarei operatii matematice.

Asadar, ne putem imagina problema ca un black box cu 3 intrari (polinomul 1, polinomul 2, operatia de efectuat) si o iesire (rezultatul operatiei). In acest sens, vom avea doua campuri de text unde utilizatorul poate introduce date care sa fie validate si procesate in obiecte de tip Polinom. Daca ambele campuri au continut valid, atunci utilizatorul poate sa apase pe un buton corespunzator unei operatii matematice, primind rezultatul intr-un camp de text separat. De asemenea, trebuie ca aplicatia sa includa un buton de curatare a input-ului, care reseteaza toate campurile de text.

Scenariul in care utilizatorul introduce doua polinoame valide si selecteaza o operatie este unul de la sine inteles: rezultatul va fi afisat in campul de text asociat rezultatului.

In cazul in care un polinom sau doua sunt invalide, utilizatorului ii va fi afisat un mesaj de eroare pentru a semnala lucrul acesta, operatia nu se va duce la bun sfarsit, si mai mult de atat, ii va fi semnalizat exact care camp de text nu contine un polinom valid, prin incercuirea campului de text cu o culoare rosie (iar celui valid cu o culoare verde).

In cazul putin probabil in care utilizatorul introduce doua polinoame valide, dar din oarece motiv, operatia nu se poate duce la bun sfarsit, se va afisa un mesaj de eroare, iar ambele campuri de text vor fi incercuite cu o culoare verde, semnaland in mod clar o eroare la nivel de logica a operatiei, ce trebuie raportata si rezolvata.

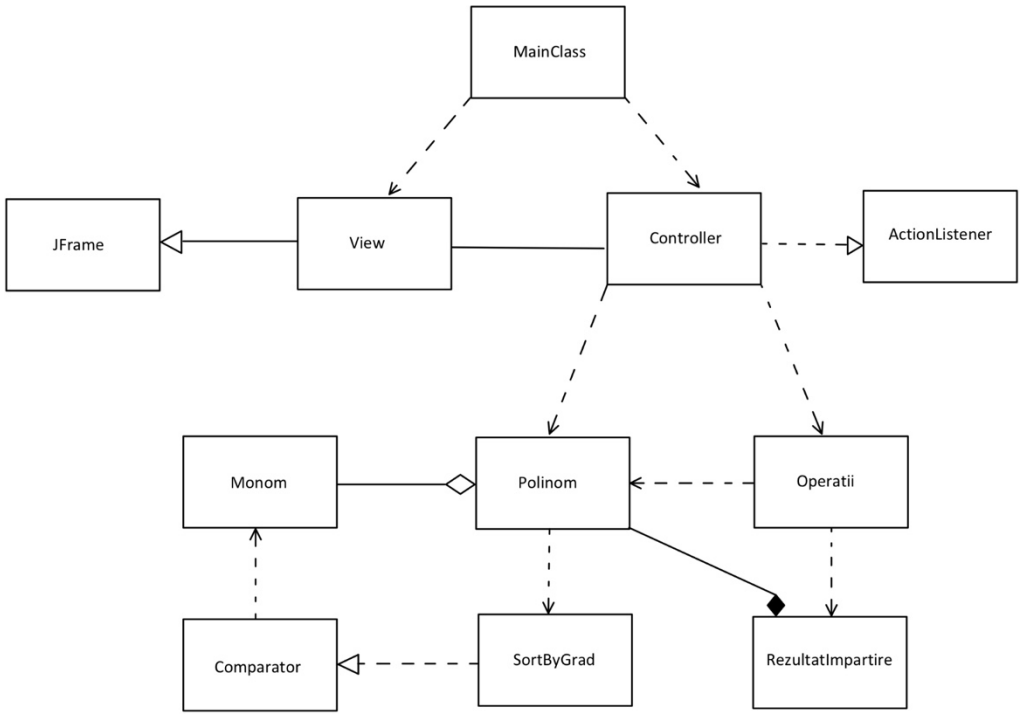
3. Proiectare

In cadrul proiectarii claselor, variabilele instanta din fiecare clasa au fost declarate cu modificatorul de acces “private”, pentru a respecta paradigma incapsularii OOP. Astfel, datele stocate intr-un obiect pot fi citite si/sau modificate doar prin intermediul metodelor de tip getter si setter.

De asemenea, am folosit mostenirea in definirea clasei View si am implementat atat interfata ActionListener, in contextul fiecarui buton si camp de text ce se regaseste pe interfata grafica, cat si interfata Comparator, pentru a putea sorta toate elementele expresiilor polinomiale.

In cadrul clasei Monom, am definit 2 variabile de tip instanta: coeficient si grad, care sunt de tipul Double si respectiv Integer. Asadar, am folosit paradigma impachetarii OOP pentru a reprezenta tipuri de date primitive in clase invelitoare ce stocheaza valori de tip double si int, pentru a ma putea folosi de metode statice implementate pentru aceste clase, in implementarea ulterioara a parsarii si a operatiilor matematice.

In cadrul proiectarii aplicatiei de calculator de polinoame, clasele definite sunt definite si relationeaza in modul prezentat in diagrama UML de mai jos.



In cadrul temei am folosit liste de tip ArrayList pentru a stoca succesiunea de monoame in obiectele de tip Polinom si pe tot parcursul operatiilor am iterat prin aceste liste folosind for each-ul specific OOP.

De asemenea, am definit o clasa RezultatImpartire care joaca rolul unei structuri de date traditionale, aceasta avand doua variabile instanta de tip Polinom, una pentru a stoca catul impartirii si a doua pentru a stoca restul.

Pe tot parcursul temei m-am folosit de o metoda esentiala in optimizarea tuturor operatiilor, chiar si a parsarii si stocarii polinomului, si anume metoda instanta *adaugaElement* din cadrul clasei Polinom, care asigura faptul ca nu vor exista mai multe monoame cu acelasi grad in interiorul expresiei polinomiale. De fiecare data cand efectuez o operatie si trebuie sa adaug monoame la expresia rezultatului, folosesc aceasta metoda.

De asemenea, am definit o metoda numita *stergeRedundant* in cadrul clasei Polinom, pentru a sterge toate monoamele cu coeficient 0, deoarece evaluarea matematica a unui monom cu coeficientul 0 ar fi chiar 0, deci nu ar influenta rezultatul in vreun fel, totodata facandu-l mai greu de interpretat.

4. Implementare

In cadrul implementarii temei am definit clasele prezentate mai jos, ale caror cele mai importante metode au fost si detaliate.

1. Clasa MainClass
   - Fara variabile instante sau de clasa
   - Metoda *main*

2. Clasa Monom
   - Variabilele instanta: Integer grad, Double coeficient; fara variabile de clasa
   - Metoda *divideMonom*
     - Folosita la operatia de impartire a doua polinoame; se defineste un nou obiect de tip Monom ce va avea gradul diferentei celor doua monoame si impartirea coeficientilor celor doua monoame drept noul coeficient
   - Metoda *toString*
     - Folosita la afisarea polinoamelor; functie avansata de prelucrare a datelor unui monom pentru a afisa un polinom in mod natural (de exemplu, monomul 5*x^0 va fi afisat drept doar 5)

3. Clasa Polinom
   - Variabila instanta: List<Monom> expresie; fara variabile de clasa
   - Metoda *adaugaTermen*
     - Folosita la fiecare operatie de modificare a expresiei polinomiale; asigura faptul ca nu vor exista mai multe monoame cu acelasi grad, pentru o foarte buna lizibilitate
   - Metoda *stergeRedundant*
     - Folosita la fiecare operatie de modificare a expresiei polinomiale; asigura faptul ca nu vor exista monoame cu coeficientul 0, pentru o foarte buna lizibilitate
   - Metoda static *parsarePolinom*
     - Folosita in cadrul clasei Controller pentru a translata textul introdus de la tastatura al utilizatorului intr-o expresie polinomiala valida; metoda complexa, admite o varietate de cai de a descrie un monom, cum ar fi:
       - 2x (sau 2X)
       - 2*x (sau 2*X)
       - 2x^1 (sau 2X^1)
       - 2*x^1 (sau 2*X^1), cu oricat de multe spatii intre coeficienti, x, exponent si operatori

4. Clasa Operatii
   - Fara variabile instante sau de clasa
   - Metodele statice ale operatiilor matematice (*adunaPolinoame, scadePolinoame, etc)*
     - Folosite la apasarea butoanelor asociate pe interfata grafica; returneaza un polinom rezultat al operatiei (cu exceptia functiei *impartirePolinoame*, care returneaza un obiect de tipul RezultatImpartire, ce contine la randul sau 2 polinoame)

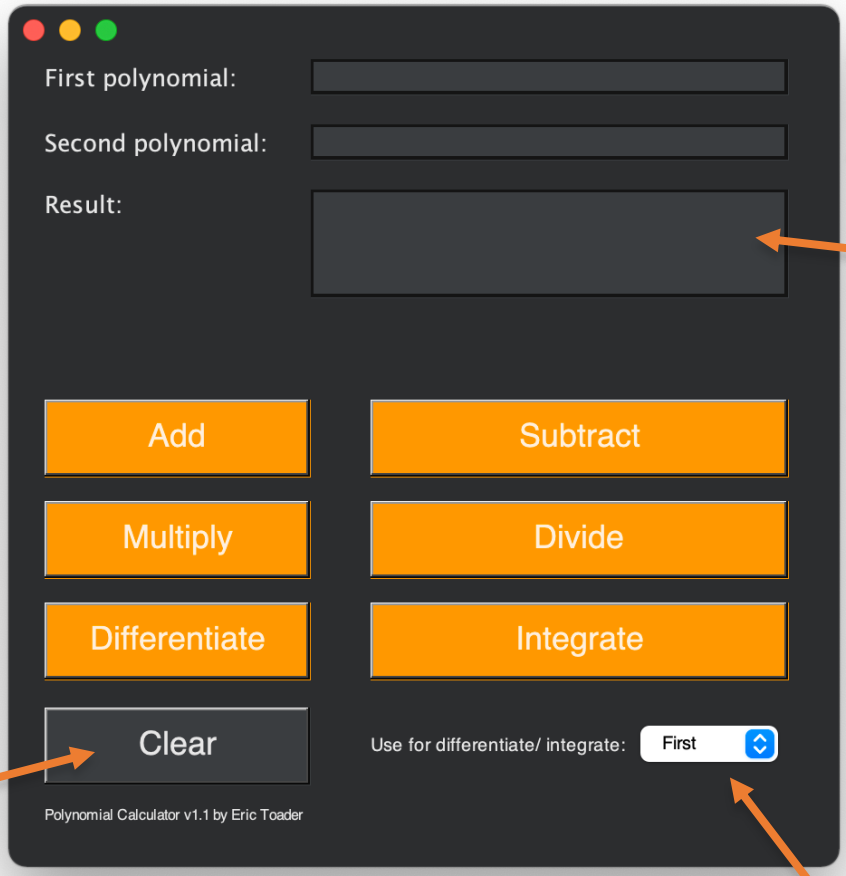
5. Clasa RezultatImpartire
   - Variabile instanta: Polinom polinomCat, Polinom polinomRest; fara variabile de clasa

6. Clasa SortByGrad
   - Implementeaza interfata *Comparator<Monom>*
   - Metoda *compare*
     - Folosita pentru a putea ordona expresia polinomiala in ordine descrescatoare a gradului, pentru o foarte buna lizibilitate

7. Clasa Controller
   - Variabila instanta: View v; fara variabile de clasa

- Subclasele asociate fiecarui buton de operatie matematica, butonului de clear si campurilor de text pentru afisarea corectitudinii datelor inserate (*AdditionListener, SubstractListener, etc)* ce implementeaza interfata ActionListener
- Metoda *displayCorectness*
    - Folosita de fiecare data cand se intercepteaza o exceptie in interiorul metodelor *actionPerformed* ale subclaselor

8. Clasa View
    - Variabile instante: butoane, campuri de text si combo-box-ul de selectare a polinomului pentru operatiile de integrare/ derivare; fara variabile de clasa
    - Metoda constructor *View()* care constituie frame-ul de afisat pe ecran, in cadrul caruia sunt amplasate butoanele si celelalte elemente interactive, impreuna cu JLabel-uri, dispuse intr-un GridBagLayout, toate stilizate folosind metoda *stylize()*, care confera aplicatiei un aspect modern si asemanator unui "Dark Mode"



Campuri text de introducere a polinoamelor

Camp text de afisare a rezultatului operatiei

Butoane corespunzatoare operatiilor matematice

Buton de clear ce reseteaza toate campurile text

Combo-box pentru selectarea polinomului dorit pentru efectuarea operatiilor de integrare/ derivare

5. Rezultate

In cadrul testarii, am folosit libraria JUnit pentru a testa corectitudinea fiecarei operatii.Inaintea fiecarui nou test, initializez variabilele instanta de tip Polinom p1 si p2 cu valorile prestabilite p1 = 5x^3 + 3x + 2 si p2 = 3x^2 + 2x

```
@BeforeEach
void setUp() {
    // p1 = 5x^3 + 3x + 2
    p1 = new Polinom();
    p1.adaugaTermen( grad: 3, coeficient: 5.0);
    p1.adaugaTermen( grad: 1, coeficient: 3.0);
    p1.adaugaTermen( grad: 0, coeficient: 2.0);

    // p2 = 3x^2 + 2x
    p2 = new Polinom();
    p2.adaugaTermen( grad: 2, coeficient: 3.0);
    p2.adaugaTermen( grad: 1, coeficient: 2.0);

}
```

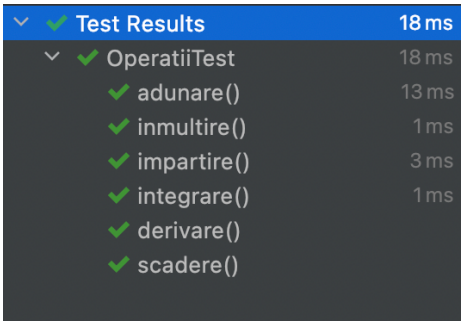
Testele pentru fiecare operatie au fost implementate comparand rezultatul operatiei in cauza cu rezultatul corect matematic, obtinut prin calculul traditional al polinoamelor. De exemplu, testarea pentru operatia de adunare a fost realizata in felul urmator:

```
@Test
void adunare() {
    Polinom result = new Polinom();
    result.adaugaTermen( grad: 3, coeficient: 5.0);
    result.adaugaTermen( grad: 2, coeficient: 3.0);
    result.adaugaTermen( grad: 1, coeficient: 5.0);
    result.adaugaTermen( grad: 0, coeficient: 2.0);

    assertTrue(Operatii.adunaPolinoame(p1,p2).toString().equalsIgnoreCase(result.toString()));
}
```

Toate celelalte functii de test au fost implementate avand in vedere aceeasi strategie, intr-un mod similar.

Toate testele efectuate asupra operatiilor matematice de calcul al polinoamelor au trecut cu success, avand asadar un total de 6 teste trecute din 6.



6. Concluzii

In urma realizarii acestei teme am deprins abilitatea de a crea o interfata grafica prietenoasa si intuitiva in Java Swing, familiarizandu-ma si mai mult cu modelul Model View Controller si libraria in sine.

De asemenea, am invatat sa folosesc GitLab si produse similare Git pentru a partaja codul si ultimele modificari efectuate, ceea ce se va dovedi a fi o unealta foarte puternica atat pentru temele si proiectele ce vor urma, dar mai ales in cadrul mediului locului de munca, unde se va prefera o abordare similara pentru stocarea si distributia codului sursa.

In urma finalizarii temei am exportat proiectul intr-o aplicatie cu extensia .jar pe care am salvat-o local pentru a o folosi ulterior, ori de cate ori voi avea nevoie, calculatorul de polinoame avand multe intrebuintari utile si practice in domeniul real.

In ceea ce priveste o posibila dezvoltare ulterioara a calculatorului de polinoame, sunt de parere ca implementarea unui istoric ar fi un pas urmator natural in realizarea unui calculator complet si modern. O posibila modalitate de implementare ar fi adaugarea unei noi variabile instanta de tip ArrayList de Istoric (o noua clasa ce contine 5 variabile instanta: Polinom 1, Polinom 2, operatia, rezultatul sub forma de Polinom, rezultatul sub forma de RezultatImpartire) la care se adauga elemente noi cu fiecare operatie efectuata cu succes de utilizator. Acest ArrayList va fi serializat si stocat local ca sa fie citit si afisat in timp real si la deschiderea aplicatiei.

De asemenea, am putea integra si un sistem de raportare de defectiuni cu ajutorul unei baze de date ce contine doua tabele (UserReports si CriticalBugs) ce contin rapoarte trimise manual de utilizatori in cazul in care ei cred ca o operatie nu este efectuata corect in cazul tabelului UserReports si respectiv rapoarte trimise automat de catre aplicatie (in conditiile in care masina ce ruleaza aplicatia are acces la internet) in cazul intampinarii unei erori neasteptate, ce contin coduri de eroare, polinoame de intrare, operatie efectuata si alte date vitale pentru testarea, diagnosticarea si rezolvarea erorilor, in cazul tabelului CriticalBugs.

7. Bibliografie

- Java Swing:
  - https://docs.oracle.com/javase/tutorial/uiswing/

- JUnit:
  - https://www.vogella.com/tutorials/JUnit/article.html

  - https://www.baeldung.com/junit-5

- Java naming conventions
  - https://google.github.io/styleguide/javaguide.html