Text

Description automatically generated

DOCUMENTAȚIE

TEMA 1-Polynomial Calculator

Buruian Cătălina

Grupa 30221

# 

**CUPRINS**

1. [Obiectivul temei 2](#_Toc97418559)

1. [Analiza problemei, modelare, scenarii, cazuri de utilizare 2](#_Toc97418560)
2. [Proiectare 2](#_Toc97418561)
3. [Implementare 2](#_Toc97418562)
4. [Rezultate 2](#_Toc97418563)
5. [Concluzii 2](#_Toc97418564)
6. [Bibliografie 2](#_Toc97418565)

**Capitolul 1:Obiectivul temei**

Obiectivul principal

Obiectivul principal al acestei teme a constat in proiectarea si implementarea unui calculator de polinoame care primeste doua polinoame si afiseaza rezultatul unei operatii selectate in prealabil de utilizator prin intermediul unei interfete grafice.

Obiective secundare

1. Analiza problemei si identificarea cerintelor necesare rezolvarii problemei ( Cap 2 )
2. Proiectarea calculatorului de polinoame ( Cap 3)

-proiectarea unei interfete grafice care face faciliteaza interactiunea cu utilizatorul

1. Implementarea calculatorului de polinoame ( Cap 4 )

-implementarea operatiilor de adunare, scadere, inmultire, impartire, derivare, integrare

1. Testarea rezultatelor ( Cap 5 )

**Capitolul 2: Analiza problemei, modelare, scenarii, cazuri de utilizare**

Cerinte functionale:

1. Calculatorul de polinoame ar trebui sa permita utilizatorilor sa insereze cele doua polinoame, adica datele de intrare
2. De asemenea, calculatorul de polinoame ar trebui sa permita utilizatorilor sa alega una din cele sase operatii matematice posibile: adunare, scadere, inmultire, impartire, derivare, integrare.
3. In urma efectuarii pasilor de mai sus si in functie de optiunile utilizatorilor, calculatorul va afisa un rezultat.

Use case1: Adunarea a doua polinoame

Actorul principal: Utilizatorul

Principalul scenariu de succes:

1. Utilizatorul insereaza doua polinoame in TextField-urile din cadrul interfetei grafice.
2. Utilizatorul alege optiunea „Add”, apasand pe butonul aferent.
3. Calculatorul realizeaza adunarea polinoamelor si afiseaza rezultatul operatiei de adunare in TextField-ul asociat rezultatului.

Scenariu alternativ: Polinoame incorecte

Utilizatorul introduce polinoame care nu sunt valide, nu respecta formatul pentru polinoame (ex.: 2\*x^2-5\*x)

Use case2: Scaderea a doua polinoame

Actorul principal: Utilizatorul

Principalul scenariu de succes:

1. Utilizatorul insereaza doua polinoame in TextField-urile din cadrul interfetei grafice.
2. Utilizatorul alege optiunea „Subtract”, apasand pe butonul aferent.
3. Calculatorul realizeaza scaderea polinoamelor si afiseaza rezultatul operatiei de scadere in TextField-ul asociat rezultatului.

Scenariu alternativ: Polinoame incorecte

Utilizatorul introduce polinoame care nu sunt valide, nu respecta formatul pentru polinoame (ex.: 2\*x^2-5\*x)

Use case3: Inmultirea a doua polinoame

Actorul principal: Utilizatorul

Principalul scenariu de succes:

1. Utilizatorul insereaza doua polinoame in TextField-urile din cadrul interfetei grafice.
2. Utilizatorul alege optiunea „Multiplicate”, apasand pe butonul aferent.
3. Calculatorul realizeaza scaderea polinoamelor si afiseaza rezultatul operatiei de inmultire in TextField-ul asociat rezultatului.

Scenariu alternativ: Polinoame incorecte

Utilizatorul introduce polinoame care nu sunt valide, nu respecta formatul pentru polinoame (ex.: 2\*x^2-5\*x)

Use case4: Impartirea a doua polinoame

Actorul principal: Utilizatorul

Principalul scenariu de succes:

1. Utilizatorul insereaza doua polinoame in TextField-urile din cadrul interfetei grafice.
2. Utilizatorul alege optiunea „Divide”, apasand pe butonul aferent.
3. Calculatorul realizeaza impartirea polinoamelor si afiseaza rezultatul operatiei de impartire in TextField-ul asociat rezultatului.

Scenariu alternativ: Polinoame incorecte

Utilizatorul introduce polinoame care nu sunt valide, nu respecta formatul pentru polinoame (ex.: 2\*x^2-5\*x)

Use case5: Derivarea unui polinom

Actorul principal: Utilizatorul

Principalul scenariu de succes:

1. Utilizatorul insereaza un polinom in primul TextField din cadrul interfetei grafice.
2. Utilizatorul alege optiunea „Derivative”, apasand pe butonul aferent.
3. Calculatorul realizeaza derivarea polinomului si afiseaza rezultatul operatiei de derivare in TextField-ul asociat rezultatului.

Scenariu alternativ: Polinoame incorecte

Utilizatorul introduce polinoame care nu sunt valide, nu respecta formatul pentru polinoame (ex.: 2\*x^2-5\*x)

Use case6: Integrarea unui polinom

Actorul principal: Utilizatorul

Principalul scenariu de succes:

1. Utilizatorul insereaza un polinom in primul TextField din cadrul interfetei grafice.
2. Utilizatorul alege optiunea „Integrate”, apasand pe butonul aferent.
3. Calculatorul realizeaza integrarea polinomului si afiseaza rezultatul operatiei de integrare in TextField-ul asociat rezultatului.

Scenariu alternativ: Polinoame incorecte

Utilizatorul introduce polinoame care nu sunt valide, nu respecta formatul pentru polinoame (ex.: 2\*x^2-5\*x)

Diagram

Description automatically generated

**Capitolul 3: Proiectarea**

**Nivelul 1: Overall system design**

**Text

Description automatically generated with medium confidence**

**Nivelul 2: Impartirea in sub-sisteme/pachete**

**Modelul architectural utilizat: MVC**

Model-view-controller (MVC) este un model arhitectural utilizat în ingineria software. Succesul modelului se datorează izolării logicii de business față de considerentele interfeței cu utilizatorul, rezultând o aplicație unde aspectul vizual sau/și nivelele inferioare ale regulilor de business sunt mai ușor de modificat, fără a afecta alte nivele.

**Model**

Această parte a controlatorului manipulează operațiunile logice și de utilizare de informație (trimisă dinainte de către rangul său superior) pentru a rezulta de o formă ușor de înțeles.

**View**

Acestui membru al familiei îi corespunde reprezentarea grafică, sau mai bine zis, exprimarea ultimei forme a datelor: interfața grafică ce interacționează cu utilizatorul final. Rolul său este de a evidenția informația obținută până ce ea ajunge la controlor.

**Controller**

Cu acest element putem controla accesul la aplicația noastră. Pot fi fișiere, scripturi (eng. scripts) sau programe, in general orice tip de informație permisă de interfață. În acest fel putem diversifica conținutul nostru de o formă dinamică și statică, în același timp.

**MVC Pattern**

Diagram

Description automatically generatedDiagram

Description automatically generated

**Nivelul 3: Impartirea in clase**

**Diagrama UML de clase**

A screenshot of a computer

Description automatically generated with medium confidence

**Clase utilizate si metodele din cadrul acestora:**

**1. Model**

Metode:

* getPolynomial
* getResult
* isNumeric

**2. View**

**3. Controller**

Metode:

* addALButton
* addActionListener
* buttons
* initController

**4. Monomial**

* divide
* toString

**5. Polynomial**

* addMonomialList
* sort
* toString

**6. Operations**

* add
* subtract
* derivate
* divide
* integrate
* multiply
* removeALLZeros

**7. SortClass**

* compare

**8. Main**

* main

**Structuri de date folosite: Liste**

O lista reprezinta o colectie ordonata de elemente. Intr-o lista sunt permise elementele duplicate. Permite un control mult mai precis asupra pozitiei unui element prin intermediul indexului acelui element.

Interfata List descrie liste de elemente indexate si are disponibile implementari precum cea utilizata in cadrul acestei teme, si anume ArrayList.

ArrayList: permite acces aleator la elementele listei; obtinearea oricarui element din lista se va face in timp constant; in schimb, adaugarea sau stergerea unui element in interiorul listei necesita deplasarea elementelor care se gasesc dupa pozitia la care se executa operatia.

**Alti algoritmi utilizati**: sortare cu metoda compare din clasa SortClass pentru a sorta polinoamele descrescator in functie de grad.

**Capitolul 4: Implementarea**

**Descrierea claselor si metodelor din cadrul claselor**

**Clasa Monomial:**-defineste un monom, adica un singur termen constituent al polinomului. Astfel, un monom contine un exponent si un coeficient. In cadrul clasei am definit, pe langa constructor, settere si gettere pentru coeficienti si exponenti si metode ajutatoare pentru operatii, precum metoda boolean isFound() si Monomial divide(Monomial monomial). Clasa contine si o functie de afisare String toString().

**Clasa Polynomial:**-defineste un polinom, adica mai multe monoame aranjate sub forma unei liste. Printre metodele importante ale acestei clase se numara metoda de afisare String toString(Polynomial p) si cea de sortare descrescatoare in functie de exponenti void sort(List<Monomial> monomialList).

**Clasa Operations:**-contine metode pentru efectuarea operatiilor de adunare, scadere, inmultire, impartire, derivare si integrare a polinoamelor.

Metoda Polynomial add(Polynomial p, Polynomial q)- declaram o lista de monoame cu ajutorul careia vom itera pe cele doua polinoame prin intermediul instructiunii foreach. Pentru fiecare polinom, parcurgem element cu element, adica fiecare monom din fiecare polinom pentru a cauta monoame cu aceeasi exponenti. Odata gasite se apeleaza setFound(true) pentru ambele monoame, si continuam prin a efectua adunarea propriu zisa: creem un nou monom cu coeficientul reprezentat de suma celor 2 coeficienti , si exponentul primului monom. Monomul rezultat este adaugat in lista de polinoame. Ulterior, monoamele care nu au fost gasite, adica nu au avut pereche ca exponent, sunt introduse si ele in lista. La final, apelez metoda removeALLZeros pentru a sterge monoamele cu coeficientii nuli si sortez lista rezultata in urma adunarii.

Exemplu adunare din interfata:

**Table

Description automatically generated**

Metoda Polynomial subtract(Polynomial p, Polynomial q)- declaram o lista de monoame cu ajutorul careia vom itera pe cele doua polinoame prin intermediul instructiunii foreach. Pentru fiecare polinom, parcurgem element cu element, adica fiecare monom din fiecare polinom pentru a cauta monoame cu aceeasi exponenti. Odata gasite se apeleaza setFound(true) pentru ambele monoame, si continuam prin a efectua scaderea propriu zisa: creem un nou monom cu coeficientul reprezentat de diferenta celor 2 coeficienti , si exponentul primului monom. Monomul rezultat este adaugat in lista de polinoame. Ulterior, monoamele care nu au fost gasite, adica nu au avut pereche ca exponent, sunt introduse si ele in lista si daca fac parte din al doilea polinom vor avea semnul “-“ in fata. La final, apelez metoda removeALLZeros pentru a sterge monoamele cu coeficientii nuli si sortez lista rezultata in urma adunarii.

Exemplu scadere din interfata:

Table

Description automatically generated

Metoda Polynomial multiply(Polynomial p, Polynomial q:- declaram o lista de monoame cu ajutorul careia vom itera pe cele doua polinoame prin intermediul instructiunii foreach. Pentru fiecare polinom, parcurgem element cu element, adica fiecare monom din fiecare polinom si le inmultim fiecare cu fiecare, inmultind coeficientii perechilor de monoame, si adunand exponentii. Lista rezultata va contine si monoame cu acelasi exponent astfel ca acestea trebuie adunate, pentru a simplifica rezultatul. Pentru aceasta, am impartit lista rezultata in 2 liste, una care contine prima jumatate a listei si cealalta care contine a doua jumatate a listei. Cu listele rezultate am construit 2 polinoame si am apelat metoda de adunare pentru cele doua polinoame. La final, am sortat polinomul rezultat.

Exemplu inmultire din interfata:

Table

Description automatically generated

Metoda Polynomial divide(Polynomial p1, Polynomial p2):-verifica initial daca polinomul cu care se imparte este diferit de 0, in caz contrar arunca o exceptie de tipul RuntimeException. Se initializeaza catul cu 0 si restul cu primul polinom. Apoi, cu un while care se executa atat timp cat restul e nenul si coeficientul termenului dominant al restului e mai mare decat cel al polinomului p2 salveaza rezultatul impartii termenului dominant al restului la termenul dominant al lui p2 in monomul t, il aduna la cat, apoi scade din rest inmultirea dintre cat si t. La final se returneaza catul impartirii.

Exemplu impartire din interfata:

Table

Description automatically generated

Metoda Polynomial derivate(Polynomial p): -parcurge polinomul p si adauga la o noua lista definita in prealabil un monom cu coeficientul egal cu inmultirea dintre coeficient si exponent si exponentul egal cu exponentul-1. La final, apelez metoda removeALLZeros pentru a sterge monoamele cu coeficientii nuli si sortez lista rezultata.

Exemplu derivare polinom din interfata:

**Table

Description automatically generated**

Metoda Polynomial integrate(Polynomial p): -parcurge polinomul p si adauga la o noua lista definita in prealabil un monom cu coeficientul egal cu impartirea dintre coeficient si exponent+1 si exponentul egal cu exponentul+1. La final, apelez metoda removeALLZeros pentru a sterge monoamele cu coeficientii nuli si sortez lista rezultata.

Exemplu integrare polinom din interfata:

Table

Description automatically generated

**Clasa Model:**- reprezinta partea care face posibila efectuarea propriu-zisa a calculelor matematice. Astfel, prin intermediul metodei String getResult(Polynomial p1, Polynomial p2, String operator) care este apelata cu doua variabile de tip Polynomial care reprezinta cele doua polinoame si un String care reprezinta numele operatii este returnat polinomul obtinut sub forma de String. Acesta polinom urmeaza sa fie afisat in interfata.

De asemenea, in cadrul acestei clase am creat si o metoda Polynomial getPolynomial(String polynomial) care primeste un String care reprezinta polinomul din interfata si face conversia la tipul Polynomial pentru a putea efectua operatii cu acesta. Aceasta metoda foloseste regex si pattern matching pentru a separa String-ul si a prelua coeficientii si exponentii fiecarui monom in parte.

**Clasa View:**- contine declaratii de butoane, text field-uri, labels, panels si alte componente necesare pentru realizarea unei interfete care sa imbunatateasca experienta utilizatorului. (Se va detalia mai incolo)

**Clasa Controller:** - aceasta clasa realizeaza conexiunea dintre interfata grafica si programul propriu-zis. Contine metode precum void addActionListener() care in momentul cand utilizatorul apasa pe butonul asociat unei operatii se preiau datele din textFielduri si se genereaza un rezultat care este scris in textFieldul rezultat. De asemenea, contine o metoda care adauga ActionListener pentru fiecare buton din interfata, cat si o metoda pentru a alege in care textField introduce utilizatorul datele de intrare.

**Clasa SortClass:** - aceasta implementeaza interfata Comparator<Monomial> cu metoda compare(Monomial o1, Monomial o2) care compara gradele celor doua monoame.

**Clasa Main**:- este clasa in care spunem programului ce sa execute. Creeaza un view, un model si un controller.

**Descrierea GUI**

**Table

Description automatically generated with medium confidence**

-in partea de sus are doua textFielduri pentru introducerea polinoamelor de intrare si un textField in care va aparea rezultatul operatiei

-polinoamele se pot introduce fie de la tastatura, fie apasand pe butoanele care reprezinta cifrele si alte caractere

-interfata contine chiar si un buton care poate sterge ce a introdus utilizatorul in textField pentru a-l lasa sa introduca alt polinom

-apasand pe butoanele specifice celor sase operatii disponibile va aparea rezultatul dorit in textFieldul rezultat

**Capitolul 5: Rezultate**

Termenul de „testare unitară” se referă la testarea individuală a unor unități separate dintr-un sistem software. În sistemele orientate pe obiecte, aceste „unități” sunt de regulă clase și metode. Uneltele de testare unitară pot înregistra testele pentru ca ele să poată fi repetate ușor mai târziu (de regulă când se schimbă o parte din sistem), astfel încât dezvoltatorul să fie convins că noile modificări nu au stricat vechea funcționalitate. Acest lucru e cunoscut sub numele de testare regresivă. Conceptele testării unitare și testării regresive sunt destul de vechi, dar popularitatea lor a crescut, după publicarea metodologiei de programare extreme și după apariția unei unelte de testare unitară pentru Java: JUnit. JUnit este un cadru de testare regresivă scris de Erich Gamma și Kent Beck.

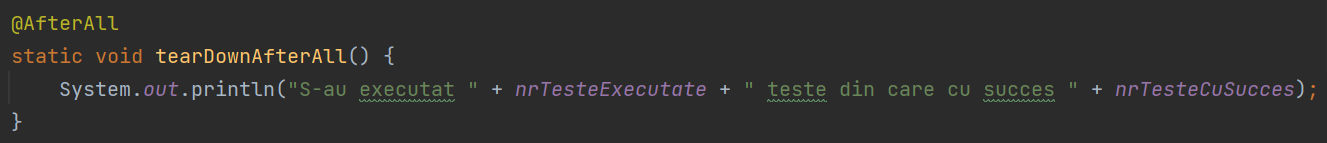
Pentru a realiza testarea am creat o clasa OperationsTest unde am creat metode de test pentru fiecare operatie in parte. Am folosit testarea parametrizata pentru a realiza teste pentru mai multe valori de intrare/iesire.

Inainte de teste am initializat modelul.

**Graphical user interface

Description automatically generated**

Dupa teste se va afisa nr de teste executate si nr de teste executate cu succes.

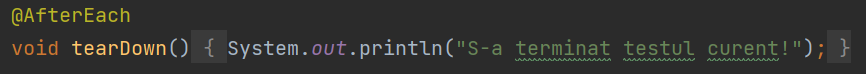


Inainte de fiecare test va creste variabila nrTesteExecutate.

Graphical user interface, text

Description automatically generated

Si dupa fiecare test se va afisa un mesaj.



Exemplu test parametrizat pentru operatia de adunare:

Text

Description automatically generated

Rezultate dupa efectuarea tuturor testelor:

Text

Description automatically generated

**Capitolul 6: Concluzii**

In concluzie, aceasta tema m-a ajutat sa aprofundez anumite aspect legate de OOP, cat sa-mi si reamintesc notiunile invatate semestrul trecut. A fost o buna oportunitate pentru mine sa ma familizarizez cu limbajul de programare Java, cu crearea de interfete grafice, cu folosirea unor patternuri arhitecturale precum MVC si cu folosirea de regular expressions.

Dezvoltari ulterioare: crearea unui calculator de polinoame cu mai multe variabile.

**Capitolul 7: Bibliografie**

1. <http://tutorials.jenkov.com/java-regex/matcher.html>
2. <https://ro.wikipedia.org/wiki/Model-view-controller>
3. <http://csfaculty.tcu.edu/comer/20803/MVCDemo.pdf>
4. <https://junit.org/junit5/docs/current/user-guide/>
5. <https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/Assertions.html>
6. <https://www.baeldung.com/parameterized-tests-junit>
7. <https://docs.oracle.com/javase/tutorial/uiswing/>