DOCUMENTATIE

TEMA 1

Polynomial Calculator

Vamvu Denisa-Elena

Grupa 30229

CUPRINS

1. Obiectivul temei

-Se va prezenta obiectivul principal al temei printr-o fraza si un tabel sau o lista cu obiectivele secundare. Obiectivele secundare reprezinta pasii care trebuie urmati

pentru indeplinirea obiectivului principal. Fiecare obiectiv secundar va fi descris si

se va indica in care capitol al documentatiei va fi detaliat.

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

-Modelare, scenarii, cazuri de utilizare Se va prezenta cadrul de cerinte functionaleformalizat si cazurile de utilizare ca si diagrame si descrieri de use-case.

Descrierile use-case-urilor se vor face sub forma unui flow-chart ori sub forma unei liste continand pasii executiei fiecarui use-case.

1. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

-Se va prezenta proiectarea OOP a aplicatiei, diagramele UML de clase si de pachete, structurile de date folosite, interfetele definite si algoritmii folositi (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator).

4. Implementare

-Se va descrie fiecare clasa cu campurile si cu metodele importante. Se va descrie implemantarea interfetei utilizator.

5. Rezultate

-Se vor prezenta scenariile pentru testare cu Junit sau alt framework de testare.

6. Concluzii

-Se vor prezenta concluziile, ce s-a invatat din tema, dezvoltari ulterioare.

7. Bibliografie

-Se vor mentiona resursele bibliografice care au fost folosite pentru dezvoltarea

temei

1. Obiectivul temei

1.1 Obiectiv principal

Propuneti, proiectati si implementati un calculator de polinoame cu o singura variabila, cu interfata grafica dedicata prin care utilizatorul poate sa introduca polinoame si sa selecteze operatia pe care o doreste a fi efectuata pe acestea:

* Adunarea a doua polinoame P1(x) + P2(x)
* Scaderea a doua polinoame P1(x) - P2(x)
* Inmultirea a doua plinoame P1(x) \* P2(x)
* Impartirea a doua polinoame P1(x) / P2(x)
* Derivarea unui polinom P1(x)’
* Integrarea unui polinom ∫ P1(x)

1.2 Obiective secundare

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Dezvoltarea de use case-uri si scenarii | Într-un sistem software un ‘use-case’ este o listă de acțiuni sau pași de eveniment care definesc în mod obișnuit interacțiunile dintre un rol ( cunoscut ca un actor în limbajul unificat de modelare ( UML ) ) și un sistem in atingerea unui obiectiv.  In computere, un scenariu este o naratiune a interactiunilor dintre roluri ( actori in UML ) si sistem. In cazul nostru rolul - un utilizator al aplicatiei. | 3 |
| Alegerea structurilor de date | Structurile de date folosite pentru a implementa cerinta propusa. | 3 |
| Impartirea pe pachete | Folosirea unui MVC ( Model – View – Controller) | 3 si 4 |
| Dezvoltarea algoritmilor | Vor fi descrise structurile de date necesare pentru atingerea obiectivului principal, schema UML ( diagrama de clase ) precum si algoritmii folositi pentru realizarea operatiilor. | 4 |
| Implementarea solutiei | Vor fi descrise fiecare clasa cu campurile si metodele importante precum si descrierea interfetei utilizator; | 4 |
| Testarea programului | Vor fi descrise cateva scenarii de testare a operatiilor pe polinoame, folosind ca instrument un Junit Test. | 5 |

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

Utilizarea programului presupune introducerea de catre utilizator a unor polinoame intr-una dintre cele doua situatii:

* In cazul in care doreste sa foloseasca o operatie ce poate fi executata pe doua polinoame, aplicatia dispune in partea de sus de 2 TextField-uri in care acestea pot fi introduse; dupa introducere, se poate selecta operatia, iar rezultatul va fi afisat intr-unul (sau doua in cazul impartirii) JTextPane-uri.
* In cazul in care doreste sa foloseasca operatiile de derivare sau integrare, care pot fi executate pe un singur polinom la un moment dat, aplicatia dispune in partea de jos de un JTextField separat pentru introducere, doua butoane dintre care poate fi selectata operatia dorita si un JTextPane pentru afisarea rezultatului .
* Formatul polinomului recunoscut de aplicatie este: aX^b +/- cX^d incluzand coeficientul 1 si gradul 0 pentru recunoastere.
* Ordinea puterilor nu este relevanta la introducerea unui polinom .
* Afisarea rezultatului va fi in ordinea descrescatoare a puterilor .

A screenshot of a cell phone

Description automatically generated

3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator) .

* Decizii de proiectare :

Polinomul introdus de catre utilizator ca si String, sub forma ceruta, este separat, cu ajutorul unui Regex (Regular Expression) in mai multe Monoame.

* Structuri de date folosite :

Am folosit in clasa Polynomial un TreeSet de monoame ca si structura de baza, ordonat descrescator dupa gradul acestora .

monomials=new TreeSet<Monomial>();

* Diagrama Use-Case :

A close up of text on a white background

Description automatically generated

* Diagrama UML de clase :

A screenshot of text

Description automatically generated

* Diagrama UML de pachete :

A screenshot of a cell phone

Description automatically generated

* Algoritmi :

La adunare,scadere,derivare si integrare, datorita proprietatilor polinoamelor, aceste operatii se pot face termen cu termen, parcurgand polinomul sau polinoamele o singura data.

Algoritmul de impartire a polinoamelor ce va fi explicat la punctul 4.

* Interfata utilizator :

Interfata utilizator a fost realizata cu Java.swing si este alcatuita din:

* Un JFrame ca si suport pentru fereastra ;
* 3 JTextFields pentru introducerea polinoamelor de catre utilizator (explicatie in cap. 2) ;
* 4 JLabels pentru a identifica fiecare field folosit ;
* 3 JTextPanes unde vor fi afisare rezultatele ;
* 6 JButtons pentru selectarea operatiei dorite ;

1. Implementare

* **Monomial**

Un monom este format dintr-un coeficient, un grad si o variabila ( ex. 2X^3 , unde 2 – coeficientul, X – variabila, iar 3 – gradul/puterea ). Monom-ul va fi in java o clasa separata care va contine ca atribute un intreg ce reprezinta puterea (gradul) si un float care reprezinta coeficientul ( acesta va fi de tip float deoarece la anumite operatii, se poate obtine o fractie ce este automat evaluata : ex ½ =0.5 ) .

private double coef;  
private int power;

Constructorul clasei **Monomial** ce primeste un parametru de tip String sub forma **aX^b** si foloseste functia **split** pentru a separa coeficientul si puterea, acestea fiind atribuite variabilelor de clasa cu ajutorul functiilor *parseInt* si *parseDouble* care alaturi de clasa invelitoare transforma un String in tipul de date necesar.

public Monomial(String s) {  
 if(s.length()>1) {  
 String[] p = s.split("[X^]");  
 this.coef = Double.*parseDouble*(p[0]);  
 this.power = Integer.*parseInt*(p[2]);  
 }  
}

O metoda importanta este **compareTo** a interfetei **Comparable** pe care clasa **Monomial** o implementeaza. Suprascrierea realizata mai jos este folosita de structura de date **TreeSet** pentru a stoca obiectele de tip **Monomial** in ordinea descrescatoare a puterilor, odata ce acestea sunt introduse in lista.

public int compareTo(Object obj){  
 if(this.power==((Monomial)obj).power)  
 return 0;  
 if(this.power<((Monomial)obj).power)  
 return 1;  
 return -1;  
}

Clasa **Monomial** mai contine metoda **toString()** suprascrisa pentru a afisa un “Monom” sub forma stabilita.

De asemenea, contine **Setters** si **Getters** pentru fiecare atribut.

Pe langa acestea, clasa **Monomial** mai contine toate operatiile necesare calculatorului de polinoame, pentru a fi utilizate in clasa **Polynomial**.

* **Polynomial**

Clasa **Polynomial** contine ca si atribut un **TreeSet** de obiecte de tip **Monomial**.

private TreeSet<Monomial> monomials;

Un Polinom, este deci format din mai multe Monoame.

Astfel, constructorul clasei **Polynomial** ce primeste ca si parametru un String si il desparte cu ajutorul functiei **spilt** in mai multe String-uri care reprezinta cate un monom fiecare, apeleaza constructorul clasei **Monomial** pentru a transforma fiecare String intr-un Obiect de tip Monomial,

introducand-ul mai apoi in lista sa de monoame .

public Polynomial(String s){  
 monomials=new TreeSet<Monomial>();  
 String replaceString=s.replace("-","+-");  
 String a[]= replaceString.split("[+]");  
  
 for (String b: a)  
 {  
 if(b.length()>0) {  
 Monomial mon = new Monomial(b);  
 this.monomials.add(mon);  
 }  
}  
}

Metoda **toString()** se regaseste suprascrisa si in clasa **Polynomial**, constant in parcurgerea listei de monoame si concatenarea rezultatelor apelurilor functiei **toString()** pentru fiecare monom in parte.

Clasa Polynomial mai contine si metoda

public int getMaximumDegree(){  
 return monomials.first().getPower();  
}

care returneaza gradul polinomului (reprezentand gradul maxim dintre monoame, implicit gradului primului monom), urmand sa fie folosit in algoritmul de impartire a doua polinoame.

* Model

Clasa **Model** reprezinta calculatorul propriu-zis de polinoame. Pe principiul pattern-ului folosit, MVC, clasa Model contine logica aplicatiei, iar in cazul de fata metodele necesare implementarii operatiilor.

Aceasta contine cele 6 metode corespunzatoare celor 6 operatii pe care aplicatia le ofera:

* + **addPolynomials** – primeste ca si parametrii doua obiecte de tip Polynomial si returneaza unul nou. Se executa in felul urmator: se parcurg ambele polinoame si se aduna monoamele care au puteri egale (folosind metoda de adunare din clasa Monomial). Apoi se parcurg pe rand ambele polinoame transmise ca si parametrii si se aduna monoamele ale caror puteri nu s-au gasit in celalalt polinom (adica nu au fost incluse in timpul primei parcurgeri). Rezultatul se returneaza.
  + **subPolynomials** – functioneaza similar metodei de adunare, singura diferenta fiind faptul ca in polinomul rezultat se trec cu semnul pe care il au doar monoamele primului polinom (pentru ca din el se scade) si cu semn opus monoamele celui de-al doilea polinom (pentru ca acesta este scazut). Se foloseste metoda de scadere din clasa Monomial.
  + **multiplyPolynomials –** din nou, primeste ca si parametrii doua obiecte de tip Polynomial si returneaza unul nou. Se parcurg ambele polinoame si se foloseste functia din clasa Monomial care inmulteste doua monoame intre ele. In acelasi timp se parcurge si polinomul rezultat, iar in caz ca puterea monomului auxiliar ce urmeaza a fi introdus a fost deja inserata (din cauza faptului ca folosim o structura de tip TreeSet nu se mai poate insera inca o data), se foloseste o functie care aduna cei doi coeficienti ai puterii curente.
  + **DividePolynomials –** primeste doua polinoame si implementeaza teorema impartirii cu rest pentru polinoame in felul urmator: cat timp gradul deimpartitului este mai mare decat gradul impartitorului se adauga intr-un polinom auxiliar rezultatul impartirii primelor 2 monoame (cele de grad maxim), primul monom din acest polinom se adauga in cat, dintr-o copie a deimpartitului se scade produsul dintre polinomul auxiliar si impartitor, iar apoi, cu ajutorul functiei **remove()** dintr-un TreeSet se elimina primul monom din polinomul auxiliar. La finalul buclei, ce a ramas in deimpartit reprezinta restul. Functia este de tip void si are 4 parametrii: deimpartit, impartitor, cat, rest.
  + **derivatePolynomial** -primeste ca si parametru un polinom, iar prin parcurgerea cu un for each a listei monoamelor din care acesta este format si apelarea functiei de derivare din clasa Monomial genereaza rezultatul.
  + **integratePolynomial** - functioneaza la fel ca si functia de derivare, apeland insa o functie de integrare a unui monom.

**GUI – Graphical User Interface**

* Partea de GUI a fost implementata cu ajutorul toolkit-ului Swing al limbajului Java si este impartita in :
* View

Clasa View are ca si atribute de clasa 6 butoane corespunzatoare operatiilor, 3

TextField-uri pentru introducerea polinoamelor, JTextPane-uri pentru afisarea rezultatelor si JLabel-uri pentru etichetarea fieldurilor.

Aceasta clasa contine metode de adaugare de ActionListenere pentru fiecare buton in parte. De asemenea, contine metode care permit afisarea de mesaje de eroare in label-uri in caz ca utilizatorul a apasat pe un buton si nu a introdus polinomul.

* Controller

Clasa Controller contine un View si un Model si leaga logica aplicatiei de interfata acesteia. Clasa mai contine si alte clase care implementeaza interfata ActionListener si se ocupa cu actiunile care urmeaza a fi efectuate dupa ce s-a apasat un anume buton.

1. Rezultate

Cu ajutorul JUnit putem testa daca operatiile sunt implementate correct. Intr-un test, cu antetul generat automat de unitatea de testare, se introduc 2 polinoame si rezultatul asteptat in urma operatiei asupra acestora. Se foloseste **assertEquals** pe toString()-urile rezultatului si al rezultatului asteptat pentru a verifica egalitatea. Un exemplu de scenariu de testare ar fi :

@Test  
void addPolynomials() {  
 Polynomial first=new Polynomial("2X^2+3X^1+5X^0");  
 Polynomial second=new Polynomial("3X^2+4X^0");  
 Polynomial result=model.addPolynomials(first,second);  
 String res="+5.0X^2+3.0X^1+9.0X^0";  
 *assertEquals*(result.toString(),res);  
}

La fel se procedeaza si pentru restul operatiilor.

1. Concluzii

In urma realizarii acestei aplicatii mi-am imbunatatit abilitatile de programare pe obiecte, am invatat sa folosesc un regex si i-am inteles utilitatea, mi-am fixat modul de lucru cu design pattern-ul MVC si am produs un calculator de polinoame functional.

Ca si dezvoltari ulterioare ale aplicatiei mele as putea enumera :

* Dezvoltarea regex-ului pentru comoditatea utilizatorului ;
* Afisarea cat mai user-friendly a polinomul ;
* Introducerea unei optiuni de CLEAR care sa elimine rezultatele ce sunt afisate la un moment dat pentru a da interfetei un aer mai “ clean ” ;
* Introducerea unei optiuni de “ dark theme “ pentru ca este populara printre aplicatii si programatori si personal o folosesc pe toate toate platformele care o ofera ;
* Introducerea unei optiuni de folosire a rezultatului returnat la un moment dat, ca si operand a unei noi operatii .

1. Bibliografie

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

<https://www.mathsisfun.com/algebra/polynomials-division-long.html>

<https://www.javatpoint.com/java-swing>