Tema 1

Calculator - operatii pe polinoame

Stefanovici Miruna Andreea

Grupa 30229

**Continut**

**1.***Obiectivul temei*

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

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

**4.***Implementare*

**5.***Rezultate*

**6.***Concluzii*

**7.***Bibliografie*

**-1-** **Obiectivul temei**

Obiectivul temei consta in implementarea operatiilor de adunare, scadere, inmultire, impartire, derivare si intregare pentru polinoame de o singura variablia, acest proces luand forma unui calculator de polinoame, cu o interfata grafica proiectata astfel incat utilizatorul poate introduce cele doua polinoame asupra carora se efectueaza operatiile ( sau un singur polinom cand doreste sa realizeze derivarea/ integrarea).

Pentru ca scopul principal sa fie atins, trebuie urmariti mai multi pasi, precum:

* analizarea potentialelor scenarii de folosire a aplicatiei de catre un utilizator, a interactiunii dintre user si program(detaliere in **Cap.2**);
* alegerea modului de proiectare, a structurilor de date folosite in cadrul acesteia, precum si dezvoltarea logicii pe care se vor baza algoritmii ce urmaresc implementarea operatiilor dorite

( detaliere in **Cap.3)**;

* alegerea modului de implementare a aplicatiei in urma unei logici stabilite si a sablonului pentru interfata grafica, precum si deciderea rolurilor fiecarei clase si conexiunilor dintre acestea

(detaliere in **Cap.3**);

* descrierea claselor prin evidentierea rolului fiecarui camp si fiecarei metode, continuand cu explicarea sablonului folosit pentru interfata grafica(detaliere **in Cap.4**);
* analizarea rezultatelor in urma implementarii si a folosirii Junit (detaliere in **Cap.5**).

**-2- Analiza problemei, modelare, scenarii, cazuri de utilizare**

Scenariul de utilizare pe care l-am vizualizat este urmatorul :

* Utilizatorul introduce cele doua polinoame.
* Utilizatorul acceseaza butonul in urma caruia se realizeaza operatia dorita.
* Rezultatul operatiei este afisat intr-un camp de text.
* In cazul alegerii operatiei de impartire, campul de text destinat rezultatelor fiecarei operatii joaca in cadrul acesteia rolul de cat, iar ce-al de-al doilea camp de text este destinat special alegerii operatiei respective, intrucat reprezinta restul impartirii.

Detalii despre formatul pe care trebuie sa il respecte polinoamele introduse se afla in Cap.4, in urma explicarii metodelor folosite. Daca acest format este respectat de catre utilizator, nu vor aparea confuzii sau erori in urma rezultatelor afisate.

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

In ceea ce priveste proiectarea aplicatiei, calculatorul pentru operatii pe polinoame este organizat conform sablonului MVC : Model-View-Controller. In cadrul laboratoarelor din semestrul trecut, ne-a fost prezentata aceasta idee, des utilizata, de a separa logica de implementare a interfetei grafice in:

* Model: algoritmii ce trebuie implementati si in urma carora se ajunge la obiectivul propus;
* View: “vedere”, vizualizare a afisajului grafic al interfetei ( buton, camp de text, label, panel, etc) ce poate interactiona cu modelul;
* Controller: mecanismul de a raspunde la cererile utilizatorului, interactionand cu vederea.

In spatele ideii de “Model” al sablonului MVC, sta logica si implementarea operatiilor de baza din matematica pe polinoame. Design-ul este unul simplist si consta in doua campuri de text pentru introducerea polinoamelor destinate operatiilor, sase butoane pentru alegerea operatiei si inca doua campuri de text pentru afisarea rezultatelor.

Pana la discutia despre operatii pe polinoame se pune mai intai problema conceptului de polinom in cadrul acestui program ( avand la baza bineinteles, conceptul matematic).

Asadar, in aplicatie, polinomul este reprezentat de clasa **Polinom**, ce are ca si camp( ca variabila instanta) structura de date TreeSet( ce nu permite duplicate si ofera un acces rapid la date), ce particular este un TreeSet de Monoame. Apare astfel conceptul de monom. Un monom este reprezentat de clasa **Monom** ce are ca si variabile instanta un intreg, cu rol de exponent si un Double, ce are rol de coeficient. Mai multe astfel de monoame alcatuiesc polinoamele pe baza carora se pot efectua operatii. De asemenea, si aceste operatii pe polinoame, au la baza operatii intre monoame. Asta inseamna ca, pentru a obtine rezultatele dorite, trebuie sa parcurgem polinoamele monom cu monom si sa realizam operatiile acestora din urma. Operatiile sunt reprezentate de clasa **OperatiiPoli**, fiecare operatie fiind descrisa printr-o metoda statica.

Pentru ca polinomul sa fie afisat conform modelului matematic, clasa Monom implementeaza interfata predefinita Comparable, astfel monoamele din fiecare polinom vor fi sortate in ordinea descrescatoare a exponentilor.

**Diagrama UML:**

**-4-Implementare**

Urmeaza descrierea ficarei clase, tinandu-se cont de logica explicata in Cap.3.

*Clasa* ***Monom***

Are ca si variabile instanta un Double ce reprezinta coeficientul monomului si un intreg ce reprezinta exponentul acestuia. Aceasta clasa reprezinta baza pe care se va implementa clasa Polinom, un polinom fiind alcatuit din mai multe monoame.

Constructorul, setter-ul si getter-ul au rolurile de a seta campurilor unui anumit obiect, anumite valori sau de a accesa valorile acestor campuri.

Pe langa acestea, sunt implementate metodele de realizare a operatiilor pe monoame, similar intre ele ca si abordare.

Metoda de adunare:

public Monom adunaM ( Monom unMonom)  
{  
 Monom aux = new Monom(0.0,0);  
 if(this.exp==unMonom.exp) {  
 aux.setCoef(this.getCoef() + unMonom.getCoef());  
 aux.setExp(exp);  
 }  
  
 return aux;  
}

* Primeste ca parametru monomul pe care vreau sa il adun la monomul pentru care apelez functia. Conditia de adunare a acestora, compunand un alt monom (aux), este ca cele doua monoame sa aiba exponentii egali. Daca aceasta conditie este indeplinita, noul monom rezultat va avea ca si coeficient suma coeficientilor celor doua monoame si ca exponent, exponentul pe care il detin acestea. Se returneaza un obiect de tipul Monom.

Metoda de scadere urmareste aceeasi structura, singura schimbare fiind aceea ca noul monom are ca si coeficient diferenta coeficientilor celor dooua monoame:

public Monom scadeM ( Monom unMonom)  
{  
 Monom aux = new Monom(0.0,0);  
 if(this.exp==unMonom.exp) {  
 aux.setCoef(this.getCoef() - unMonom.getCoef());  
 aux.setExp(exp);  
 }  
  
 return aux;  
}

Metodele de inmultire si impartire nu mai necesita verificarea egalitatii exponentilor, dar structura este similara : se returneaza un nou obiect Monom care va avea coeficientul si exponentul schimbati in functie de regulile matematica : la inmultire se inmultest coeficientii si se aduna exponentii, iar la impartire se impart coeficientii si se scad exponentii:

public Monom inmultesteM ( Monom unMonom)  
{  
 Monom aux = new Monom(0.0,0);  
 aux.setCoef(this.getCoef()\*unMonom.getCoef());  
 aux.setExp(this.getExp()+unMonom.getExp());  
  
 return aux;  
}  
public Monom imparteM ( Monom unMonom)  
{  
 Monom aux = new Monom(0.0,0);  
 aux.setCoef(this.getCoef()/unMonom.getCoef());  
 aux.setExp(this.getExp()-unMonom.getExp());  
  
 return aux;  
}

La derivare si integrare se tine cont de asemenea de cazurile matematice. De exemplu, in cadrul derivarii, daca exponentul monomului este 0 inseamna ca si coeficientul este setat la 0 ( derivarea unei constante), iar daca este mai mare ca 0 ( aici poate fi inclus si cazul in care exponentul este 1 pe care eu l-am tratat separat din motive de organizare a ideilor), coeficientul devine produsul dintre el insusi si exponentul, iar exponentul se scade cu 1:

public Monom deriveazaM ( Monom unMonom)  
{  
 Monom aux = new Monom(0.0,0);  
 if(unMonom.getExp()==0) {  
 aux.setExp(0);  
 aux.setCoef((double) 0);  
 }  
 else  
 if(unMonom.getExp()==1)  
 {  
 aux.setExp(0);  
 aux.setCoef(unMonom.getCoef());  
 }  
 else  
 if(unMonom.getExp()>1)  
 {  
 aux.setCoef(unMonom.getCoef()\*unMonom.getExp());  
 aux.setExp(unMonom.getExp()-1);  
 }  
  
 return aux;  
}

De asemenea, la integrare coeficientul devine raportul dintre el insusi si valoarea exponentului adunata cu 1, iar exponentul creste cu 1:

public Monom integreazaM ( Monom unMonom)  
{  
 Monom aux = new Monom(0.0,0);  
 if(unMonom.getExp()==0) {  
 aux.setExp(1);  
 aux.setCoef(unMonom.getCoef());  
 }  
 else  
 if(unMonom.getExp()>=1)  
 {  
 aux.setCoef(unMonom.getCoef()/(unMonom.getExp()+1));  
 aux.setExp(unMonom.getExp()+1);  
 }  
  
 return aux;  
  
  
}

Am adaugat, in mod necesar, functia compareTo ( deoarece se implementeaza interfata Comparable) ce decide modul de sortare al monoamelor in polinom ( descrescator, in functie de exponent) si functia toString() in care sunt tratate toate cazurile de afisare a unui monom ce va ajuta la implementarea functiei toStringPol() din Polinom. La monomul cu gradul cel mai mare, ce va fi afisat la inceput, va aparea si semnul “+” in cazul coeficientilor pozitivi.

*Clasa* ***Polinom***

Contine o singura variabila instanta si anume structura de date TreeSet<Monom> in care vor fi introduse monoamele.

Exista un constructor fara parametrii care va fi de folos pentru operatii, cand nu dorim sa asociem un String.

Constructorul cu parametrii are ca parametru un String ce va fi introdus de catre utilizator in campul de text destinat. Pentru a transforma String-ul in polinom se vor folosi expresii regulate (REGEX) si functii predefinite pe baza acestora, cu ajutorul pachetelor formate din biblioteci de clase : *java.util.regex.Matcher*, *java.util.regex.Pattern*. O expresie regex este un sir de caractere care definesc un sablon de cautare. De obicei, acest sablon este apoi utilizat de catre algoritmii de cautare pe siruri pentru operatiuni de cautare. Pentru un monom, am folosit expresia **([+-]?[^-+]+)**, stabilita ca Pattern.

Prin atribuirea Matcher **matcher = pattern.matcher(s)**, unde s este String-ul primit ca parametru la constructor, am stabilit unde se vor cauta stringurile de forma Pattern-ului.Cu functia apelata de care acest matcher, **matcher.find()**, caut toate string-urile care respecta forma unui monom, iar pentru fiecare potrivire, folosesc o alta expresie regex in care descriu toate formele pe care le poate lua monomul respectiv: coeficient-X-exponent(aici intra si cazul in care coficientul este -1), X-exponent ( aici intra doar cazul in care coeficientul este 1),X-simplu(intra doar cazul in care coeficientru este 1), coeficient-X si coeficient-simplu ( in cazul unei constante). Stiu, este o organizare complicata si cel mai probabil se putea alege o functie generala, care sa se potriveasca pe toate cazurile, dar chiar si asa am reusit sa gasesc o logica ce poate sa duca la obtinerea corecta a polinomului introdus. In continuare, cu ajutorul metodei group(int group) din Matcher Class pot accesa String-urile ce se traduc prin coeficienti ai monomului ce se incadreaza ca String intr-una din cazurile enumerate mai sus. De exemplu, pentru cazul coeficient-X-exponent, accesez matcherM.group(1) si matcherM.group(2), celelalte grupuri ce apartin celorlalte formate de monom fiind nule. Am reusit sa descopar ce group() ii apartine fiecarui format cu ajutorul site-ului regex101.com . Avand toate informatiile despre un monom, il creez, ii setez informatiile, dupa care il adaug in structura TreeSet.

Alte metode sunt:

-getPol(): returneaza un obiect de tip TreeSet<Monom>, adica returneaza singura variabila instanta a clasei;

-getAlfa(): returneaza un obiect de tip Monom, ce reprezinta monomul din polinomul respective, cu gradul cel mai mare, aflat prin efectiva aflare a exponentului maxim dintr-o parcurgere a polinomului ( aceasta functie va fi utila la operatie de impartire);

-adaugaMon(): adauga un monom nou in polinom;

-toStringPol(): afiseaza polinomul folosindu-se de functia de afisare a fiecarui monom ( concatenare de String-uri).

*Clasa* ***OperatiiPoli***

Contine o serie de metode statice, fiecare metoda reprezentand o operatie.

Metoda de adunare **adunaP(Polinom p1, Polinom p2)** primeste ca parametrii doua polinoame si returneaza un obiect de tip Polinom. Se creeaza in interiorul metodei un nou obiect de tip Polinom: ***adunati***.Se parcurg ambele polinoame, iar cand gasim doua monoame cu acelasi exponent, in *adunati* adaugam monomul rezultat de apelarea functiei de adunare a doua monoame. Dupa, este necesara o parcurgere a ambelor polinoame pentru a verifica daca mai exista si alte monoame unice(adica nu si-au gasit perechea in celalalt monom si astfel n-au fost adaugate la polinomul final). Cand se gasesc, se adauga la polinomul final.Noul polinom va fi returnat.

Metoda de scadere scadeP**(Polinom p1, Polinom p2)** respecta aceeasi idee de implementare, cu exceptia faptului ca va fi invocata operatia de scadere de monoame, iar la parcurgerea celui de-al doilea polinom, vor fi adaugate monoamele unice cu semn schimbat pentru a respecta operatia matematica.Ce are in plus, este o a treia parcurgere, cea a polinomului nou rezultat ***scazuti*** in care se cauta monoamele cu coeficientul diferit de 0, pentru a fi eliminate, pas necesar pentru realizarea algoritmului de impartire de mai tarziu. Astfel, monoamele cu coeficientul diferit de 0 sunt adaugate intr-un nou Polinom **scazutiNew**, care va fi returnat.

Metodele **deriveazaP(Polinom p1)** si **integreazaP(Polinom p2)** primesc cate un singur parametru, un obiect Polinom, care este parcurs o singura data, fiecare monom fiind derivate cu operatia de derivare a monoamelor si adaugat la un nou obiect de tip Polinom ce va fi returnat.

In metoda **inmultesteP** sunt parcurse cele doua polinoame, fiecare monom din p1, inmultindu-se cu fiecare monom din p2, iar rezultatul fiind adaugat la un nou Polinom ***inmultiti***. Insa, in acest polinom pot exista monoame care desi au coeficienti diferiti, pot avea exponenti egali si de aceea, acestea trebuie adunate. Cand gasesc doua monoame cu acelasi exponenet, realizez o suma a coeficientilor lor, creez un nou obiect de tip monom care va avea ca si coeficient aceasta suma si ca si exponent, exponentul pe care il au toate aceste monoame. Monomul il voi adauga intr-un nou obiect de tip Polinom pe care il voi returna.

Pentru metoda **imparteP(Polinom p1, Polinom p2, Polinom cat, Polinom rest)** am adaptat algoritmul “long division” binecunoscut din matematica.Primeste ca parametrii polinoamele care trebuie impartite, un polinom care va reprezenta restul si unul care va reprezenta catul, obiecte de tip polinom declarate in prealabil si alocate inainte de apelarea functiei. Fac a copie a deimpartitului, deoarece acesta va devein la final restul, deci va fi modificat.Cu un while fac urmatoarele operatii cat timp gradul dempartitorului( adica monomul cu gradul cel mai mare obtinut prin functia getAlfa(), grad aflat prin functia getExp() ) este mai mare decat cel al impartitorului reprezentat de cel de-al doilea polinom. Inr-un polinom ajutator *help* stochez monomul rezultat prin impartirea monomului de grad maxim din deimpartit la monomul de grad maxim din impartitor.Rezultatul va fi adaugat in polinomul care reprezinta catul, dupa care din deimpartit scadem rezultatul inmultirii dintre help si impartitor. Astfel, *copie* devine noul deimpartit si se continua pe acelasi principiu impartirile dintre monoame.*Help* va tine mereu noul rezultat al impartirii dintre monoame. Se ajunge in momentul in care copie devine 0 sau un polinom de grad mai mic decat impartitorul, adica restul impartirii.

*Clasa* ***PolView+****Clasa* ***PolController***

Am utilizat pachetele ce contin elemente de grafica java.awt, javax.swing.

**PolView** reprezinta “Vedera” aplicatiei. Sunt descrise in constructorul clasei componentele grafice : butoane, campuri de text, label-uri, panel-uri precum si situarea acestora in fereastra rezultata.

Pentru fiecare buton sunt adaugati “ascultatori”, apelati cand utilizatorul interactioneaza cu interfata, fapt ce provoaca un eveniment. Asadar, cand se da click pe buton, se invocate metoda **actionPerformed()** definite in clasa obiectului ascultator, careia ii este transmis ca parametru un obiect ActionEvent.Clasele obiectelor ascultatoare sunt definite in **PolController,** care in constructor adauga la Vedere acesti ascultatori in urma carora la apasarea butoanelor se genereaza evenimente.

Aici evenimentele inseamna operatiile pe polinoame.Fiecare metoda actionPerformed corespunde cate unei operatii. Se preiau datele introduse de catre utilizator, cele doua polinoame sau un polinom si in functie de buton, se efectueaza actiunea.

**-5-Rezultate**

Am folosit testarea cu JUnit (JUnit4), implementand o clasa ce continte 6 teste, adica un test pentru fiecare operatie.

Fiecare metoda trebuie sa aiba inainte de definire, adnotatia @Test. In urma structurii fiecarei operatii, in fiecare metoda de test se vor crea doua polinoame sau un polinom, se va efectua in alt polinom operatia, iar rezultatul va fi transpus intr-un sir de caractere cu ajutorul metodei toStringPol() si comparat, cu ajutorul instructiunii **assertEquals,** cu un String asteptat ca rezultat. Daca sunt egale cele doua String-uri, insemana ca operatia a reusit, altfel testul va afisa FAIL.

**-6-Concluzii**

In urma acestei teme cred ca mi-am exersat intr-un mod benefic abilitatea de a scrie cod. Desi tot programul poate fi imbunatatit considerabil, consider ca am atins obiectivele propuse.

Ulterior ii mai pot fi adaugate functii speciale cum ar fi verificarea formatului polinomului introdus cu o mai mare strictete sau adaugarea functiei de a inlocui variabila dintr-un polinom cu o constata pentru a afla valoarea unei functii de variabila X.

**-7-Bibliografie**

1.Laboratoarele de GUI din semestrul trecut

2.https://www.stackoverflow.com

3.https://www.wikipedia.org