**Capitolul 1.Obiectivul temei**

Obiectivul principal al lucrarii este testarea operatiilor pe polinoame de o singura variabila cu coeficienti intregi. Se discuta despre operatiile de : adunare , scadere ,derivare , integrare , impartire si produs intre polinoame cu o reprezentare aleasa ulterior.

**1.1.Obiectivele secundare sunt adresate in urmatoarele capitole :**

**Cap.2 : Analiza problemei si identificarea cazurilor de utilizare**

Cadrul de cerinte functionale e realizat de scaderea,adunarea,impartirea,inmultirea,derivarea si integrarea polinoamelor . Se vor prezenta descrieri de use-case sub forma unor flow-charts , respectiv sub forma unei liste cu pasii executiei fiecarui use-case.

**Cap.3** : **Proiectarea diagramelor UML , a algoritmilor si intefetelor utilizator**

Se vor prezenta decizii legate de proiectarea claselor , structurilor de date, interfetelor utilizate in descrierea problemei , precum si a relatiile dintre acestea , organizarea in pachete precum si diagramele de clase si pachete utile in modelarea OOP a operatiilor pe polinoame.

**Cap.4 : Implementarea propriu-zisa**

Prezentarea campurilor , constructorilor , metodelor importante care realizeaza corespondenta OOP-operatii matematice traditionale intre polinoame.Tot aici se va prezenta interfata cu utilizatorul si cateva exemple de utilizare.

**Cap5. : Rezultatele reliefate de teste Junit**

Se prezinta scenariile pentru testare a diverselor operatii prin clase si metode de testare adecvate cu Junit. Se testeaza printr-o serie de operatii validitatea oferita de implementarea metodelor corespunzatoare operatiilor.

**Cap6. : Concluzii**

In acest capitol , se discuta concluzii , precum si dezvoltari posibile ulterioare.

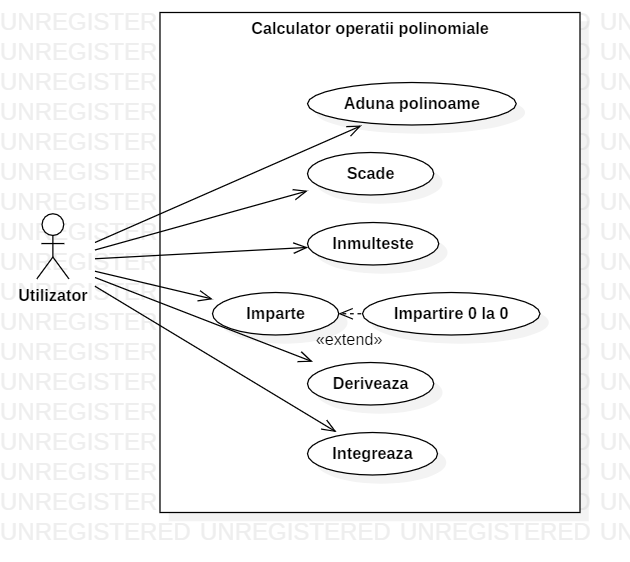
**Cap7. : Bibliografie**

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

Se prezinta in continuare functionalitatile oferite de aplicatie sub forma unei diagrame ce prezinta principalele cazuri de utilizare .

Diagrama contine 7 cazuri de utilizare dintre care 6 vor corespunde unor scenarii favorabile iar unul corespunde cazului de impartire 0 la 0,caz care apare doar la introducerea respectiva a datelor.

Scaderea este de fapt o adunare intre doua polinoame,in care al doilea intervine cu semnul schimbat,deci descrierea celor doua cazuri de utilizare se va rezuma la unul singur.

****

Nota: Cele 6 use case-uri prezina in principiu aceeasi pasi pe care utilizatorul ii parcurge pentru a ajunge la rezultatul dorit, toate avand ca secventa alternativa de utilizare neintroducerea de date in unul din campurile de care are nevoie operatia.

Se prezinta cazul adunarii, celelate fiind similare:

**• Use Case: Aduna polinoame**

**• Primary Actor: Utilizatorul aplicatiei**

**• Main Success Scenario:**

**1.**Utilizatorul este ghidat de interfata grafica pentru selectarea operatiei.

**2.** Utilizatorul introduce datele de care operatia de adunare are nevoie(2 polinoame ) .

**3.**Datele sunt verificate.

**4**.Se proceseaza rezultatul adunarii.

**5.**Sunt afisate eventualele mesaje legate de operatie.

**• Use Case: Aduna polinoame**

**• Primary Actor: Utilizatorul aplicatiei**

**• Alternative Sequences:**

1. **Nu au fost introduce date**

**1.**Se afiseaza un mesaj corespunzator

**2.**Utilizatorul e ghidat spre a introduce din nou datele.

Pentru impartirea polinoamelor ,exista cazul de nedeterminare la impartire 0/0 care poate fi rezumat in urmatoarea diagrama de utilizare care o extinde pe cea de impartire(doar unele cazuri ale intrarilor corespund acestui caz si de aceea nu avem incluziune).

**• Use Case: Impartire 0 la 0**

**• Primary Actor: Utilizatorul aplicatiei**

**• Alternative Sequences:**

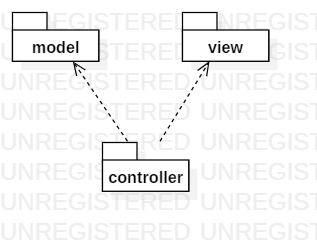
1. **Datele introduse corespund unei impartiri de forma 0/0**

**1.**Se afiseaza un mesaj corespunzator de importanta mare

**2.**Utilizatorul e ghidat spre a introduce din nou datele.

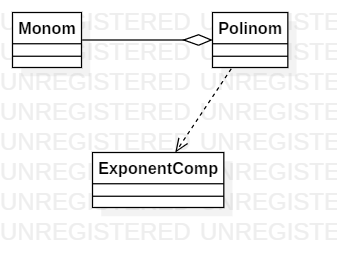
**Capitolul 3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)**

Proiectul e structurat in 4 pachete : 1 pachet care continte testele Junit si 3 pachete care prezinta arhitectura Model-View-Controller a proiectului. Se poate observa dependenta pachetului controller de model si view:



**Pachetul Model** contine urmatoarele clase necesare modelarii conceptului de polinom ( Monom , Polinom).

**Clasa Polinom** contine ca structura de date o lista de monoame (ArrayList) care pentru sortare foloseste comparatorul din acelasi pachet: ExponentComp (compara monoamele pentru o ordonare descrescatoare dupa exponent). Orice operatie care se rasfrange asupra unui polinom , de fapt se realizeaza asupra fiecarui monom din lista de monoame de aceea clasa Monom se afla in relatie de agregare si nu de compozitie cu clasa Polinom (Spunem ca un polinom agrega mai multe monoame) asa cum rezulta din diagrama de clase pentru Model:

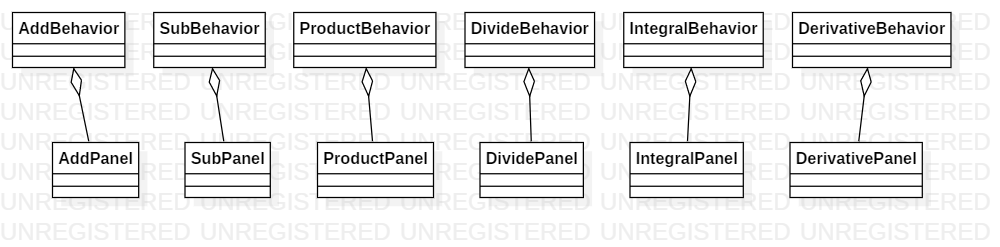


**Pachetul View** contine partea vizuala a interfetei grafice si e construita din 9 clase : 6 sunt mostenite din JPanel si corespund fiecarei operatii cu polinoame . Fiecare JPanel e activ o singura data in functie de operatia selectata. De exemplu : ulizatorul selecteaza operatia de “Scadere polinoame” din meniu , atunci celelate panouri nu vor fi active , ci doar panoul curent operatiei de scadere; o clasa reprezinta suportul pe care se afiseaza individual panourile de operatii si meniul de selectie al operatiilor ; o clasa care reprezinta frame-ul interfetei utilizator si care face legatura dintre suportul de panouri si o clasa dedicata afisarii de mesaje (de ghidare,de eroare,etc.)

Diagrama de clase corespunzatoare claselor din pachetul view:



**Pachetul controller** ataseaza posibilitati de interactiune cu panourile de operatii prezentate mai sus , deci exista 6 clase care stabilesc ce ar trebui sa se intample la interactiunea cu panourile(in cazul nostru : butoanele interfetei ) .Implementarea aleasa prezinta diagrama de clase cu legaturi de agregare (un “Behavior” are un “Panel” dar acesta din urma poate exista si in absenta comportamentului) , asa cum urmeaza:



**Capitolul 4.Implementare**

Urmeaza detalii legate de implementarea celor mai importante clase , deoarece cateva prezinta similaritati . Cele 6 clase mostenite din JPanel: AddPanel, SubPanel, ProductPanel, DividePanel, IntegralPanel si DerivativePanel au clase corespondente care le modeleaza comportamentul ,asa cum arata diagrama de clase din pachetul controller ,insa toate functioneaza dupa acelasi principiu .Acestea au intr-o mare masura aceeasi implementare (toate preiau date si initiaza actiuni la apasari de butoane,diferentele ar consta in faptul ca la derivare si integrare avem un singur camp unde se pot introduce datele),de aceea se va explica doar implementarea pentru 1 JPanel cu clasa de comportament asociata.

**Clasa Polinom(cea mai importanta)**

**Variabilele instanta ale clasei:**

private String polinom;

private ArrayList<Monom> monomList;

private int grade;

Atributul “polinom” reprezinta reprezentarea sub forma de String a datelor introduse de utilizator pentru modelarea unui polinom . “monomList” este o lista de monoame ( clasa explicata ulterior) in care se tine evidenta monoamelor componente din care e compus polinomul. Variabila instanta “grade” se refera la gradul polinomului (gradul maxim dintre gradele monoamelor). Toate acestea sunt actualizate la operarea asupra polinomului.

public Polinom(String polinom) {

this.polinom = polinom;

monomList = new ArrayList<Monom>();

if (!polinom.equals("") && !polinom.equals("0"))

process(polinom);

normalize();

grade = monomList.get(0).getExp();

else if(polinom.equals("0")) {

Monom a=new Monom("0");

grade=0;

monomList.add(a);

}

}

**Constructorul** clasei primeste un String pe care il proceseaza in felul urmator: Se testeaza daca acesta nu e vid si nu e string-ul “0” ( cazul 0 se proceseaza diferit datorita celorlalte metode);in aces caz se apeleaza metoda “process” care imparte String-ul in monoame si actualizeaza lista de monoame a clasei. Unoeri e nevoie de normalizarea polinomului introdus : eliminarea monoamelor vide , adunarea monoamelor de exponenti egali ,etc. ; acest lucru se realizeaza cu ajutorul metodei “normalize”. Cum spuneam, cazul polinomului vid(“0”) se trateaza separat si fara apelul metodei normalize (care sterge din lista de monoame pe cele vide si astfel ar sterge inclusiv singurul monom din care e format polinomul).

**Metoda de procesare a polinomului**

private void(process(String raw) {

Pattern p = Pattern.*compile*(

"((-?\\d+(?=x))?(-?[x])(\\^(-?\\d+))?)|((-?)[x])|(-?\\d+)");

raw = raw.replaceAll("\\s+","");

Matcher m = p.matcher(raw);

while (m.find()) {

if (!m.group(0).equals("")) {

if (!isZero(m.group(0))) {

Monom mon = new Monom(m.group(0));

monomList.add(mon);

}

}

}

}

Procesarea de monoame se realizeaza cu ajutorul unei expresii regulate care imparte String-ul primit ca parametru in felul urmator: Se cauta monoame de forma ; -? Inseamna ca monomul poate sau nu sa fie cu semn ; [\d](file:///\\d)+ monomul poate avea un coeficient format din mai multe cifre , si poate sa aiba sau nu putere “^” , acest lucru e marcat prin formarea a 2 grupuri delimitate de () si prin operatorul sau “|”.

public void normalize() {

Collections.*sort*(this.monomList, new ExponentComp());

int i = 0;

while (i < monomList.size() - 1) {

if (monomList.get(i).getCoeff() == 0) monomList.remove(i);

} else {

if (monomList.get(i).getExp() == monomList.get(i+1).getExp()) {

monomList.get(i + 1).add(monomList.get(i)); monomList.remove(i);

} else

i++;

}

}

...

}

Functia ordoneaza monoamele polinomului in ordinea descrescatoare a exponentilor , dupa care parcurge monoamele consecutive 2 cate 2 astfel : verifica daca monomul a si monomul b consecutive au acelasi exponent , daca da : b=b+a si a se sterge din lista , daca nu , parcurgerea merge mai departe. Metoda totodata sterge din lista de monoame rezultatele partiale la adunare care au dus la monoame vide( exemplu : ) O serie de multe astfel de perechi ar genera in afisarea rezultatului o serie de adunari si scaderi cu multe zerouri ceea ce nu se doreste.

**Metoda care aduna 2 polinoame**

Obs: Metoda “sub” care scade 2 polinoame e o adunare in care termenul 2 intervine cu semn schimbat.

public String sum(String pol2) {

try {

int n1 = Integer.*parseInt*(this.polinom); int n2 = Integer.*parseInt*(pol2);

return String.*valueOf*(n1 + n2);

} catch (NumberFormatException e) {

Polinom p2 = new Polinom(pol2);

String res = this.polinom + p2.polinom;

Polinom p = new Polinom(res);

if (p.polinom.charAt(0) == '+')

p.polinom = p.polinom.substring(1);

return p.polinom;

}

}

Un apel ar metodei ar putea fi :

Polinom pol1=new Polinom(“2x^4-5x+2”);

String tmp=pol1.sum(p2);

Implementarea aleasa nu prezinta un grad mare de integrabilitate insa pentru scopul metodei (de a actualiza campul de rezultat al interfetei -un JTextField, cu rezultatul adunarii polinoamelor ) e mai eficient . Cum spuneam mai sus, fiecare polinom are o reprezentare in String .Metoda concateneaza String-ul polinomului “this” cu String-ul primit ca argument si formeaza un nou polinom care se proceseaza si normalizeaza.Se returneaza String-ul reprezentant al noului obiect de tip Polinom.

**Metoda care imparte 2 polinoame**

public String div(String pol2) {

...

Polinom d = new Polinom(pol2);

String q = "";

Polinom r = new Polinom(this.polinom);

String t = "";

Monom leadingD = new Monom(d.monomList.get(0).getRawString());

while (!r.polinom.equals("0") && r.grade >= d.grade)

t = (r.monomList.get(0).div(leadingD)).getRawString();

q = q + t;

Polinom use=new Polinom(q);

q=use.polinom;

Polinom tmp = new Polinom(t);

tmp.changeSign();

String y=r.sum(tmp.times(d.polinom));

if(y.equals("0")) {

r.polinom="0";

}else

r = new Polinom(y);

...

}

Metoda modeleaza impartirea traditionala a polinoamelor. Monomul “leadingD” reprezinta monomul de grad maxim din impartitor. Metoda imparte monomul de grad maxim din deimpartitul curent r la monomul “leadingD”.Rezultatul se memoreaza in variabila “t”. Pasul urmator(2) il constituie adunarea dintre deimpartitul curent si produsul dintre “t” luat cu semn schimbat si impartitor. Se repeta pasul (2) pana cand r devine nul sau gradul deimpartitului ajunge mai mic decat gradul impartitorului.

}

**Clasa Monom**

Aceasta clasa executa in principal operatiile de adunare, scadere nu(e tot o adunare dar cu semn schimbat),impartire,inmultire,integrare,derivare pe 2 monoame.

private String rawString;

private int coeff;

private int exp;

public Monom(String raw) {

this.rawString = raw;

process(raw);

if (rawString.charAt(0) != '-' && rawString.charAt(0) != '+') {

rawString = "+" + rawString;

}

}

Variabilele instanta “coeff”,”exp” vor preluea prin apelul metodei “process” din constructor echivalentele in intregi ale coeficientilor si exponentilor din monomul trimis ca String. Acest lucru se realizeaza prin apelul a alte 2 metode din interiorul metodei “process”: “getCoeff” si “getExponent” care examineaza String-ul trimis ca argument , stabileste in ce caz se afla(cu coeficient 0,fara coeficient,cu putere negativa,fara putere,etc.) si returneaza conversiile la intregi .

Urmeaza prezentarea celor mai importante metode ale clasei Monom:

Nota:Metoda “updateRaw” actualizeaza String-ul reprezentant al monomului ori de cate ori un coeficient sau un exponent se schimba dupa o operatie.

public void add(Monom a) {

this.coeff = a.coeff + this.coeff;

updateRaw(coeff, exp);

}

Metoda aduna coeficientii a doua monoame (this si argumentul a).

public void changeSign() {

coeff = -coeff;

updateRaw(coeff, exp);

}

Metoda utila la scadere.

public void der() {

if (exp == 0) {

coeff = 0;

else {

coeff = coeff \* exp;// (ax^b)'=abx^b-1

exp--;

}

updateRaw(coeff, exp);

}

Se realizeaza ( )’= ab

Obs: Daca exponentul e 0 , monomul reprezinta doar un numar , iar derivata unui numar e 0;

public void itgr() {

exp++;

coeff = coeff / exp;// integrala(ax^b)=(a/b+1)x^(b+1)

updateRaw(coeff, exp);

}

Se realizeaza =

public Monom times(Monom t) {

int co, exp;

co = this.coeff \* t.coeff;

exp = this.exp + t.exp;

Monom temp = new Monom("3x^2");

temp.updateRaw(co, exp);

return temp;

}

Inmultirea monoamelor inseamna inmultirea coeficientilor si adunare exponentilor(chiar daca nu sunt egali) ceea ce se si realizeaza.

public Monom div(Monom t) {

int co, exp;

co = this.coeff / t.coeff;

exp = this.exp - t.exp;

Monom temp = new Monom("3x^2");

temp.updateRaw(co, exp);

return temp;

}

Impartirea monoamelor inseamna impartirea coeficientilor si scaderea exponentilor(chiar daca nu sunt egali) ceea ce se si realizeaza .

**Interfata utilizator**

Interfata e formata dintr-un JFrame (“MainGui”), si 2 JPanel (“CustomMonitor”- clasa care doar afiseaza un mesaj intr-un JTextField dar care e static si public ceea ce il face accesibil din celelalte clase atunci cand e nevoie sa afisam un mesaj ; si ”SplitPane” ).

Panoul “SplitPane” e construit din 2 parti , frontiera dintre cele 2 parti fiind ajustabila:

* **Partea stanga: Meniul de operatii**

private String[] ops = { "Adunare", "Scadere", "Inmultire", "Impartire", "Derivare", "Integrare" };

...

opList = new JList<String>(ops);

...

opScrollPane = new JScrollPane(opList);

...

* **Partea dreapta : Panoul operatiei curente** (se instantiaza un panou ori de cate ori selectam o operatie , vor fi 6 panouri care vor fi afisate pe rand in aceeasi locatie )

private JPanel currentOpPanel;

...

protected void updatePanel(String name) {

switch (name) {

case "Adunare":

AddPanel ap=new AddPanel();

new AddBehavior(ap);

currentOpPanel = ap;

break;

case "Scadere":

SubPanel sp=new SubPanel();

new SubBehavior(sp);

currentOpPanel = sp;

break;

case "Inmultire":

ProductPanel pp=new ProductPanel();

new ProductBehavior(pp);

currentOpPanel = pp;

break;

...

splitPane.setRightComponent(currentOpPanel);

Se observa cum partea drepata a componentei “splitPane” devine un panou nou la fiecare operatie noua selectata din meniu.

**Capitolul 5. Rezultatele testelor Junit**

Se propune urmatorul tipar de clase de test parametrizate pentru fiecare din operatiile implementate pe polinoame: o clasa de test la o operatie (ex: clasa ParameterizedAdd testeaza metoda “sum” a clasei polinom printr-o colectie de date, clasa ParameterizedSub testeaza metoda “sub” prin alta colectie de date,etc.). Aceasta colectie furnizeaza datele de intrare si datele asteptate la iesire in urma unei operatii. Validarea operatiei respective se face prin metode de test (adnotate cu @Test) .

De exemplu: Aceasta e clasa care testeaza metoda “sum” a clasei Polinom.

*@RunWith*(Parameterized.class)

public class ParameterizedAdd

*@Parameters*(name="{index}: testAdd({0} + {1}={2})")

public static Collection<Object[]>data(){

return Arrays.*asList*(new Object[][] {

{"1","2","3"},

{"x^2+5x+2","-9x+x^4-x^-1-2","x^4+x^2-4x-x^-1"},

{"0+2x-88x^3","2x^5+10x","2x^5-88x^3+12x"}

});

}

Datele de intrare sunt primele 2 String-uri din colectia “data” si rezultatul asteptat e reprezentat de al treilea String.

*@Test*

public void test\_AddPolynomials() {

Polinom pol1=new Polinom(p1);

String tmp=pol1.sum(p2);

*assertEquals*(tmp,p3);

}

Datele sunt preluate in metoda adnotata cu test din clasa test corespunzatoare , se preiau datele de intrare(cele 2 string-uri in cazul adunarii) si se compara cu al treilea din colectie cu ajutorul metodei assertEquals din clasa Assert importata.

Toate aceste clase de test sunt rulate de un TestRunner.

**Capitolul 6. Concluzii**

Subiectul proiectului este complex , si prezinta multe posibilitati de implementare insa varianta aleasa a adus urmatoarele cunostiinte dobandite : realizarea de diagrame de pachete, diagrame use case , metode de eficientizare asupra operatiilor cu polinoame respectand paradigma OOP, realizarea de clase test parametrizate si cel mai important : realizarea corespunzatoare a unui proiect Java de la proiectare pana la implementare ,testare si crearea unei documentatii.

Proiectul poate fi dezvoltat in multe directii , dar propunerea adusa ar fi realizarea altor module care pot folosi sau duce la o forma polinomiala : liniarizarea unor sisteme (polinomul lui Taylor), aflarea zerourile si a punctelor critice a unei functii polinomiale , stabilirea functiilor de transfer a sistemelor invariante in timp, metode de aproximare cu ajutorul polinoamelor,polinoame de mai multe variabile ,etc.

**Capitolul 7:Bibliografie**

Interfata cu utilizatorul:-componenta JSplitPane : <https://docs.oracle.com/javase/tutorial/uiswing/components/splitpane.html>

Impartirea polinoamelor(Pseudo-code) :

<https://en.wikipedia.org/wiki/Polynomial_long_division#Pseudo-code>