

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**CATEDRA CALCULATOARE**

Calculator de polinoame

Documentație

Iacob Liviu

Grupa 302210 | An 2 semestrul 2

Cuprins

1. Obiectivul temei.

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

3. Proiectare.

4. Implementare.

5. Rezultate.

6. Concluzii.

7.Bibliografie.

1. Obiectivul temei:

Obiectivul principal al temei a fost sa propnem, sa proiectam sis a implementam un calculator de polinoame. Mai exact, un calculator de 2 polinoame care face urmatoarele operatii: adunare, scadere, inmultire, impartire, derivare si integrare. Ca orice fel de calculator, acesta opereaza pe date de intrare ( 2 polinoame ) si comunica cu utilizatorul prin date de iesire ( polinomul rezultat dupa efectuarea operatiilor ). De asemenea, pe langa partea de implementat operatiile, se poate gasi si o interfata grafica “User friendly”, care asigura usurinta in utilizare de catre orice utilizator.

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

Un polinom este vazut ca si o multitudine de monoame, care sunt la randul lor o pereche de variabile, adica coeficient si exponent (ax^b, unde a este coeficientul care poate fi pozitiv sau negative si b este exponentul). Modelul care ne-a fost sugerat in implementarea acestui proiect a fost sa consideram polinomul ca un ArrayList de monoame, fapt care a usurat mult partea de modelare a claselor. Pentru a usura partea de interactiune cu utilizatorul, in interfata grafica se gasesc 2 TextField-uri(TF) in care utilizatorul poate introduce direct polinoamele de forma aceasta: 1x^5+2x^3-4x^1-2x^0. Pentru a fi posibil acest lucru, am folosit Regex-uri pentru a separa din string-ul dat ca si imput monoamele, si a identifica la randul lor exponentii si coeficientii pentru fiecare monom. Aceasta metoda desi este putin mai complexa decat limitarea utilizatorului la un polinom de un anumit grad finit, sau obligarea utilizatorului sa scrie coeficientul 0 acolo unde nu exista monom cu gradul respective, are limitarile si dezavantajele ei, iar daca string-ul nu este de forma din exeplu vor aparea erori.

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

Am decis sa proiectez clasele astfel incat sa fie cat mai intuitiv si mai lizibil codul. Clasa Monomial are 2 atribute numite sugestiv coeficient si exponent, metode standart de get, set pentru aceste atribute cat si constructori (cu si fara parametri). Tot in clasa Monomial se mai gasesc si metode de adunare (aduna), scadere (scade), si inmutire (inmulteste) care au ca parametri 2 obiecte de tip Monomial, si care efectueaza operatiile respective pe obiecte din clasa Monomial. Din cauza esteticii, este implementata si o metoda toString(), care ajuta la afisarea cat mai frumoasa a monoamelor pe ecran.

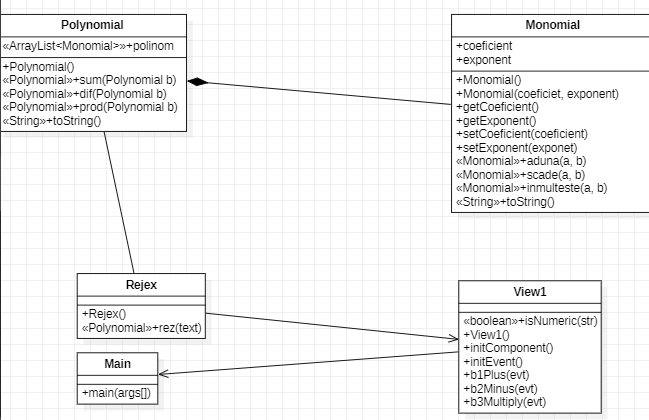
Clasa Polynomial are ca si atribut doar o colectie de obiecte apartinand clasei Monomial, implementata ca ArrayList, cum ne-a fost sugerat. Aici se vor gasi atat constructrul fara parametri si metoda toString, cat si metode de adunare (sum), scadere (dif), si inmultire (prod), care au ca parametri un obiect de tipul Polynomial.

In clasa View1 este implementata interfata grafica. Aici sunt grupate elementele in metode atat pentru pentru initializarea lor, cat si pentru atribuire coordonatelor si dimensiunilor. De asemenea, tot aici se afla si metodele pentru initializarea evenimentelor si descrierea acestora.

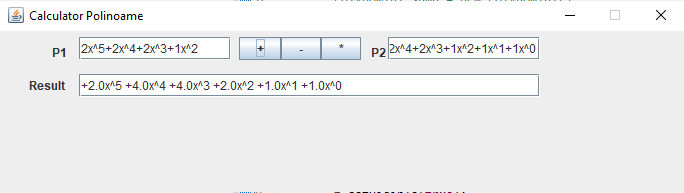
Clasa Main a fost proiectata sa fie cat mai simpla. Am lasat o bucata de cod relevanta pentru testarea initiala a metodelor implementate, care a ramas comentata. Aici se apeleaza doar constructorul de View1 si se seteaza ca aceasta fereastra sa fie vizibila.

Clasa Rejex a fost proiectata ca sa preia din string-urile de intrare datele utile si sa ignore restul textului. Aici este definit tiparul dupa care ar trebui sa fie introduce polinoamele. Acest tipar este folosit in inerfata grafica pentru a face verificarea. Daca nu este introdus corespunzator, va genera erori.

**Diagrama UML:**



**Interfata grafica:**

****

**Algoritmii folositi** pentru implementarea metodelor sunt proiectati dupa algoritmul de interclasare pentru operatiile de adunare si scadere, urmarind sa adauge in lista noului polinom toate monoamele pana cand gaseste cate un monom din ambele polinoame, care au grade egale, moment in care face adunarea/scaderea coeficientilor. In cazul inmultirii, se face o parcurgere cu 2 for-uri, parcurgand pentru fiecare element din polinomul a toate elementele din polinomul b si inmultind coeficientii si adunand exponentii.

A fost folosit un singur pachet (package) in proiectarea acestui proiect.

1. Implementare.

**Clase**:

1. Clasa Monomial:

Are ca atribute 2 variabile de tip double reprezentand coeficientul si exponentul monomului. Metodele de aduna si scade sunt gandite doar pentru monoame cu exponenti egali, verificare este facuta in metodele aferente din clas Polynomial. Metoda inmulteste realizeaza inmultirea a 2 monoame si are ca rezultat alt monom. Metoda toString asigura afisarea monoamelor.

Metode:

**public** Monomial () {}

**public** Monomial(**double** coeficient, **double** exponent)

{

**this**.coeficient=coeficient;

**this**.exponent=exponent;

}

**public** **double** getCoeficient()

{

**return** coeficient;

}

**public** **double** getExponent()

{

**return** exponent;

}

**public** **void** setExponent(**double** exponent)

{

**this**.exponent=exponent;

}

**public** **void** setCoeficient(**double** coeficient)

{

**this**.coeficient=coeficient;

}

**public** Monomial aduna(Monomial a, Monomial b)

{

Monomial x= **new** Monomial();

x.setCoeficient(a.coeficient + b.coeficient);

x.setExponent(a.exponent);

**return** x;

}

**public** Monomial scade (Monomial a, Monomial b){

Monomial x= **new** Monomial();

x.setCoeficient(a.coeficient - b.coeficient);

x.setExponent(a.exponent);

**return** x;

}

**public** Monomial inmulteste (Monomial a, Monomial b)

{

Monomial x= **new** Monomial();

x.setCoeficient(a.coeficient \* b.coeficient);

x.setExponent(a.exponent + b.exponent);

**return** x;

}

**public** String toString() {

**if** (coeficient >= 0)

**return** "+" + coeficient + "x^" + (**int**)exponent + " ";

**else**

**return** coeficient + "x^" + (**int**)exponent + " ";

}

1. Clasa Polynomial:

Are ca singur atribut o colectie de Monoame. Ca si metode implementate sunt sum si dif care parcurg listele de monoame ale celor 2 polinoame in regim de interclasare, iar in momentul gasirii a 2 monoame cu grade egale se efectueaza metodele din Monomial de adunare/scadere. In plus, la dif, daca polinomul scazator iar cel descazut nu, toti coeficientii acestuia sunt trecuti cu semn schimbat in polinomul rezultat. In functia de inmultire(care a fost initial implementata cu for(i…..) iar apoi transpusa prin stilui for each) se declara un polinom p1 la fiecare schimbare a monomului din primul polinom, astfel incat in p1 se vor retine rezultatele partiale obtinute prin inmultirea unui element din primul polinom cu toate celelalte din al 2 lea polinom. La sfarsitul fiecarei iteratii din for-ul exterior, la p2 (polinomul final) se aduna p1, adica rezultatul partial. Metoda de toString este implementata pentru a afisa estetic polinomul, facand apel la metoda toString din clasa Monomial.

**Metode:**

**public** Polynomial () { }

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_adunare\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**public** Polynomial sum(Polynomial b) {

Polynomial rez= **new** Polynomial();

Monomial aux=**new** Monomial(0, 0);

Monomial x=**new** Monomial(0, 0);

Monomial y=**new** Monomial(0, 0);

**int** n = **this**.polinom.size();

**int** m = b.polinom.size();

**int** i=0;

**int** j=0;

**while**(i < n && j < m)

{

x=**this**.polinom.get(i);

y=b.polinom.get(j);

**if** (x.getExponent() > y.getExponent()) {

rez.polinom.add(x);

i++;

}

**else** {

**if** (x.getExponent() < y.getExponent()) {

rez.polinom.add(y);

j++;

}

**else** {

aux=aux.aduna(x, y);

rez.polinom.add(aux);

i++;

j++;

}

}

}

**while**(i < n) {

x=**this**.polinom.get(i);

rez.polinom.add(x);

i++;

}

**while** (j < m)

{

y=b.polinom.get(j);

rez.polinom.add(y);

j++;

}

**return** rez;

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_scadere\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**public** Polynomial dif(Polynomial b) {

Polynomial rez= **new** Polynomial();

Monomial aux=**new** Monomial(0, 0);

Monomial x=**new** Monomial(0, 0);

Monomial y=**new** Monomial(0, 0);

**int** n = **this**.polinom.size();

**int** m = b.polinom.size();

**int** i=0;

**int** j=0;

**while**(i < n && j < m)

{

x=**this**.polinom.get(i);

y=b.polinom.get(j);

**if** (x.getExponent() > y.getExponent()) {

rez.polinom.add(x);

i++;

}

**else** {

**if** (x.getExponent() < y.getExponent()) {

y.coeficient=-y.getCoeficient();

rez.polinom.add(y);

j++;

}

**else** {

aux=aux.scade(x, y);

rez.polinom.add(aux);

i++;

j++;

}

}

}

**while**(i < n) {

x=**this**.polinom.get(i);

rez.polinom.add(x);

i++;

}

**while** (j < m)

{

y=b.polinom.get(j);

y.coeficient=-y.getCoeficient();

rez.polinom.add(y);

j++;

}

**return** rez;

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_inmultire\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**public** Polynomial prod(Polynomial b) {

Monomial aux=**new** Monomial(0, 0);

Polynomial p2 =**new** Polynomial();

/\*

Monomial x=new Monomial(0, 0);

Monomial y=new Monomial(0, 0);

int n = this.polinom.size();

int m = b.polinom.size();

for (int i=0; i<n; i++)

{

x=this.polinom.get(i);

Polynomial p1 =new Polynomial();

for (int j=0; j<m; j++)

{

y=b.polinom.get(j);

aux=aux.inmulteste(x,y);

p1.polinom.add(aux);

}

p2=p2.sum(p1);

}

\*/

**for** (Monomial i:**this**.polinom)

{

Polynomial p1 =**new** Polynomial();

**for** (Monomial j:b.polinom)

{

aux=aux.inmulteste(i,j);

p1.polinom.add(aux);

}

p2=p2.sum(p1);

}

**return** p2;

}

**public** String toString() {

String s = "";

**for**(**int** i = 0; i < polinom.size(); i++)

s += polinom.get(i).toString();

**return** s;

}

1. Clasa View1:

Extinde JFrame, faprt datorita caruia am fost nevoit sa fac import la anumite biblioteci:

**import** javax.swing.\*;

**import** java.awt.Point;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.event.WindowAdapter;

**import** java.awt.event.WindowEvent;

Nu are atribute. La inceputul clasei se afla declaratiile obiectelor folosite pentru a implementa interfata grafica: 3 JButton ( +, -, \*), 3 JTextFiel-uri (2 pentru introducerea polinoamelor ca date de intrare si unul pentru afisarea rezultatului in urma executarii operatiei corespunzatoare butonului apasat, si 3 JLabel(“P1”, “P2”, “Result”).

Constructorul fara parametri seteaza tritlul feresteri si o deschide la o anumita locatie si dimensiune pe ecran. De asemenea, aici se defineste si fereastra ca ne fiind “resizable”.

Metoda initComponent() grupeaza toate initializarile de butoane, textfield si label si le asigneaza coordinate si dimensiuni aferente, iar initEvent() adauga actionListener pe fiecare buton in parte. Urmeaza cate o metoda pentru fiecare buton in care se descrie comportamentul programului in urma apasarii lor.

**Metode:**

**public** **static** **boolean** isNumeric(String str) {

**return** str.matches("[+-]\*\\d\*\\.?\\d+");

}

**public** View1(){

setTitle("Calculator Polinoame");

setSize(700, 200);

setLocation(**new** Point(100, 200));

setResizable(**false**);

initComponent();

initEvent();

}

**private** **void** initComponent(){

getContentPane().setLayout(**null**);

b1.setBounds(239, 6, 42, 23);

getContentPane().add(b1);

b2.setBounds(281, 6, 40, 23);

getContentPane().add(b2);

b3.setBounds(321, 6, 40, 23);

getContentPane().add(b3);

tf.setBounds(79, 6, 152, 23);

getContentPane().add(tf);

tf1.setBounds(388, 6, 152, 23);

getContentPane().add(tf1);

tf2.setBounds(79, 43, 461, 23);

getContentPane().add(tf2);

l.setBounds(52, 10, 17, 23);

getContentPane().add(l);

l1.setBounds(371, 10, 17, 23);

getContentPane().add(l1);

l2.setBounds(29, 43, 40, 23);

getContentPane().add(l2);

}

**private** **void** initEvent() {

**this**.addWindowListener(**new** WindowAdapter() {

**public** **void** windowClosing(WindowEvent e) {

System.*exit*(1);

}

});

b1.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

b1Plus(e);

}

});

b2.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

b2Minus(e);

}

});

b3.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

b3Multiply(e);

}

});

}

**private** **void** b1Plus(ActionEvent evt) {

Rejex ba = **new** Rejex();

Polynomial a = **new** Polynomial();

a = ba.rez(tf.getText());

Polynomial b = **new** Polynomial();

b = ba.rez(tf1.getText());

tf2.setText(a.sum(b).toString());

}

**private** **void** b2Minus(ActionEvent evt){

Rejex ba = **new** Rejex();

Polynomial a = **new** Polynomial();

a = ba.rez(tf.getText());

Polynomial b = **new** Polynomial();

b = ba.rez(tf1.getText());

tf2.setText(a.dif(b).toString());

}

**private** **void** b3Multiply(ActionEvent evt) {

Rejex ba = **new** Rejex();

Polynomial a = **new** Polynomial();

a = ba.rez(tf.getText());

Polynomial b = **new** Polynomial();

b = ba.rez(tf1.getText());

tf2.setText(a.prod(b).toString());

}

1. Clasa Rejex

Aici se defineste cum se parseaza datele de intrare de tip String in date utile pentru proiectul nostru. Aceasta clasa a fost inspirata de pe stackoverflow.com si din prezentarea facuta de profesoara de la laborator. Se gaseste si aici un constructor fara parametri.

**Metode:**

**public** Rejex() {

}

**public** Polynomial rez(String text) {

String monomialFormat = "([+-]?[\\d\\.]\*[a-zA-Z]?\\^?\\d\*)";

String monomialPartsFormat = "([+-]?[\\d\\.]\*)([a-zA-Z]?)\\^?(\\d\*)";

Pattern p1 = Pattern.*compile*(monomialFormat);

Matcher m1 = p1.matcher(text);

Polynomial a = **new** Polynomial();

**while** (!m1.hitEnd()) {

m1.find();

Pattern p2 = Pattern.*compile*(monomialPartsFormat);

Matcher m2 = p2.matcher(m1.group());

**double** coefficient = 0;

**double** exponent = 0;

**if** (m2.find()) {

coefficient = Double.*valueOf*(m2.group(1));

exponent = Double.*valueOf*(m2.group(3));

}

Monomial b = **new** Monomial(coefficient, exponent);

a.polinom.add(b);

}

**return** a;

}

1. Clasa Main

Aceasta clasa a fost creata initial doar pentru testarea metodelor de adunare si scadere din clasa Polynomial inainde de implementarea interfetei grafice, motiv pentru care am lasat in cometarii testele facute. La momentul de fata are ca unic scop declararea unui obiect care e instanta a clasei View1, care deschide fereastra de unde se intampla toate interactiunile cu utilizatorul.

5.Rezultate.

Nu am folosit nici un program extern pentru testarea proectului, am folosit doar date de intrare date de la tastatura in repetate randuri si incercand sa acopar cat mai multe cazuri posibile. In varianta finala a proiectului nu au mai aparut erori sau greseli in calcule. Programul a fost testat atat inainte de interfata grafica si regex-uri, cat si dupa, fiind rezolvate toate problemele gasite cu exceptia erorii in care nu sunt introduse in formatul corespunzator datele de intrare.

6. Concluzii.

In concluzie, acest proiect a intarit si aprofundat cunostiintele mele despre limbajul Java, dar mai ales la capitolul interfete grafice, unde am fost nevoit sa intreb in toate partile ca sa imi astup lipsurile si lacunele de anul trecut. Ca si dezvoltari ulterioare, se pot implementa si celelalte operatii din laborator (imartire, derivare, integrare). De asemenea, se poate implementa cu un obiectiv mai inlat, acela de a servi ca unealta elevilor de clasele a 12 a pentru verificarea temelor sau exercitiilor la matematica sau pentru Pregatirea de Bacalaureat sau admitere.

7.Bibliografie.

Principala sursa de inspiratie si documentare a fost stackoverflow.com, consolidate cu ajutorul si comunicarea cu colegii de la facultate sau de la alte facultati, cat si cu “Sfantul google”.