DOCUMENTATIE

TEMA 1

NUME STUDENT: Dragotoniu Corina Mădălina

GRUPA: 30224

# CUPRINS

[1. Obiectivul temei 3](file:///C:\Users\40770\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 4](file:///C:\Users\40770\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297886)

[3. Proiectare 5](file:///C:\Users\40770\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297887)

[4. Implementare 6](file:///C:\Users\40770\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297888)

[5. Rezultate 10](file:///C:\Users\40770\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297889)

[6. Concluzii 11](file:///C:\Users\40770\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297890)

[7. Bibliografie 11](file:///C:\Users\40770\Downloads\PT2021-2022_Documentation_Template%20(1).doc#_Toc95297891)

# Obiectivul temei

Obiectivul temei este de a implementa un calculator polinomial cu interfață grafică prin intermediul căruia utilizatorul poate introduce polinoame pentru care se vor efectua diverse operații precum adunare, scădere, împărțire, înmulțire.

Obiectivele secundare ale acestei teme sunt:

Crearea unei clase Monom în care sunt stocate valorile necesare unui monom

Crearea unei clase Polinom în care este stocată o listă de monoame

Crearea unei interfețe grafice usor de utilizat și avantajoasă pentru implementare

Organizarea aplicației folosind o arhitectură de tip MVC

Implementarea operațiilor pe polinoame, adunare, scădere etc.

Testarea operațiilor pe polinoame folosind testarea unitară – Junit

Obiectivele secundare menționare anterior vor fi dezvoltate în capitolul 4.

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

Conform cerintei, Calculatorul de polinoame trebuie sa implementeze operatiile elementare asupra polinoamelor. Astfel, am putea sa incepem analiza problemei din acest punct, identificand in primul rand datele de intrare si de iesire.

La o analiza mai atenta a problemei, avem ca si date de intrare:

* Primul polinom
* Al doilea polinom
* Operatia

Ca si date de iesire, vom obtine:

* Rezultatul principal al operatiei, respective suma,diferenta produsul sau catul
* Rezultat secundar, respective restul impartirii polinoamelor

Scenariul principal de utilizare al calculatorului este pentru efectuarea operațiilor de adunare, scădere, înmulțire, împărțire, unde vor fi necesare doua polinoame care se vor introduce prin interfața grafică. Ele vor fi pasate Controllerului care va folosi Modelul pentru a realiza operațiile și apoi va afișa rezultatul în câmpurile specifice din interfață.

Operația va fi selectată de către utilizator prin apăsarea unor butoane specifice, iar polinoamele vor fi introduse de către acesta folosind tastatura de la calculator.

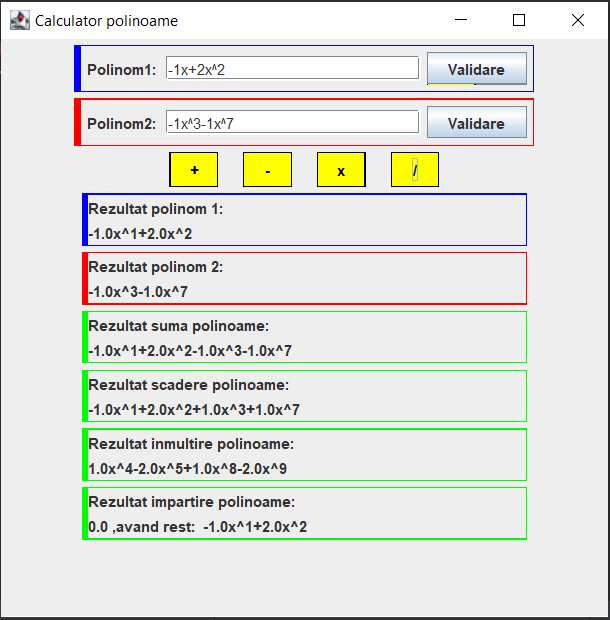
De asemenea, cand introducem polinoamele de la tastatura, adesea pot aparea situatii neasteptate, datorate modului incomplet de scriere a polinomului. De aceea, am decis sa adaug un buton de validare pentru fiecare polinom, care va executa o comanda de afisare ordonata a polinomului daca acesta a fost introdus in mod corect de la tastatura, sau afisarea unui mesaj corespunzator, in caz de esec.

# 3. Proiectare

Pentru realizarea problemei am folosit arhitectura Model-View-Controller. View-ul cuprinde strict toate elementele din interfata, este acea parte din program cu care utilizatorul intra in contact direct. Controller-ul este cel care face legătura intre View și Model. Acesta este cel care ia polinoamele introduse de către utilizator, le prelucrează pentru a corespunde modelului de date folosit, apelează operațiile din model și preia rezultatul din model, pe care-l transmite în interfață.Cuprinde, asadar, toate actiunile realizate de butoanele descrise in View. Modelul cuprinde toate structurile de date folosite si operatiile care se pot realiza cu acestea.

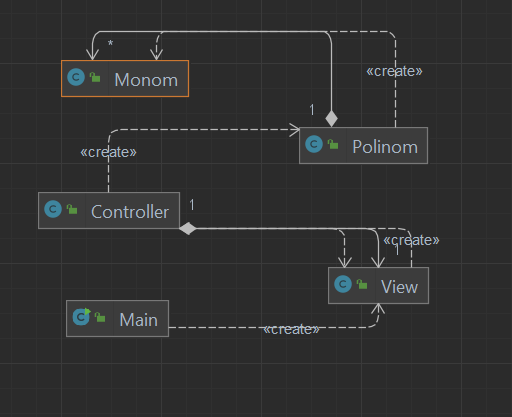
In locul unei singure clase Model, am decis sa utilizez clasele Monom și Polinom. Mai multe moname compun un polinom,, deci în mod clar clasa Polinom cuprinde o listă de monoame. Atributele pricipale ale clasei Monom sunt coeficientul și gradul, iar al clasei Polinom este ArrayList-ul de obiecte de tip Monom. Pentru fiecare clasa am scris mai multi constructori potriviți cazurilor în care avem nevoie să creăm noi polinoame rezultat.

Interfața cuprinde un grup de label-uri pentru afisarea diferitelor rezultate, 2 TextField-uri pentru introducerea polinoamelor, 2 butoane pentru validare si 4 butoane pentru operatiile specifice polinoamelor.



Calculatorul de Polinoame conține 4 clase de bază și o clasă Main ce le instanțiază.

* Clasa View– cu ajutorul ei se creează interfața grafică folosită în tot proiectul. Tot aici se instanțiază și ActionListener. Ei au rolul de a surprinde eventualele acțiuni ale utilizatorilor prin intermediul evenimentelor. Tocmai de aceea, la nivelul câmpurilor de text și a butoanelor există câte un ascultător separat.
* Clasa Controller – această clasă face legatura intre View si Model, realizand diverse actiuni in functie de ActionListener.
* Clasa Monom – clasa fundamentală a proiectului ce definește structura de monom, entitate ce conține doar un coeficient (întreg), o putere și, opțional, un String pentru a printa rapid monomul. Tot aici, se află și operațiile fundamentale: +, –, \*, / .
* Clasa Polinom – permite crearea polinoamelor ca liste de monoame, entități și mai fundamentale, dar conține și partea sortare a acestora. În acest fel se tratează unul din cazurile de utilizare mai deosebite, când utilizatorul nu respectă în totalitate formatul default al scrierii unui polinom.



# 4. Implementare

Clasa View

Conține un Jframe pe care se construiește întreaga interfață.

mainFrame = **new** JFrame("Calculator polinoame");

mainFrame.setSize(500,500);

Se poate remarca încă de la început ca avem o singura zona principala in interfață, cea de control,reprezentată de câmpurile și butoanele prin care utilizatorul introduce datele și poate să vadă primele rezultate concrete asupra polinoamelor.

Aici există câteva segmente de cod de aceeași natură, care se repetă cu mici variații. De exemplu, cele sașe zone în care se afișează informațiile despre polinoame și rezultatul operațiilor elementare sunt structural identice, atâta doar că își procură informația din alte locuri.

cLabel1 = **new** JLabel("", JLabel.***LEFT***);

cLabel1.setPreferredSize(**new** Dimension(350,20));

cLabel1.setText("Polinomul 1");

sLabel1 = **new** JLabel("",JLabel.***LEFT***);

sLabel1.setVerticalTextPosition(JLabel.***TOP***);

sLabel1.setPreferredSize(**new** Dimension(350,20));

JPanel cm = **new** JPanel();

cm.setLayout(**new** GridLayout(2,1));

cm.add(cLabel1);

cm.add(sLabel1);

În partea de mai sus se poate remarca structura de bază a unei zone de afișare a datelor despre polinoame. Cele două label-uri se află într-un panel cu layout-ul de tip grid, deoarece la redimensionarea ecranului, lățimea se modifică și vrem o oarecare repoziționare a câmpurilor.

Metoda showEvents() conține cu precădere ascultătorii, care sunt utilizați mai departe în clasa Controller.

plus.setActionCommand("plus");

plus.addActionListener(**new** Controller(**this**));

Clasa Controller

Este clasa care gestionează toate acțiunile venite din partea utilizatorilor și le corelează cu funcționalitățile preexistente. Tocmai de aceea, această clasă implementează interfața **ActionListener** și implicit metoda **actionPerformed()**. Există astfel o singură metodă care gestionează toate evenimentele venite din partea de View și asta fiindcă nu avem nevoie de un paralelism la nivelul tratării acțiunilor – în sensul că utilizatorul nu va executa două sau mai multe acțiuni în mod simutan.

Una din cele mai importante trăsături a clasei este că ea conține tot timpul două String-uri. Utilizatorul va introduce tot timpul informația sub formă de text în cele 2 JTextField-uri. Informația este neschimbată atâta timp cât utilizatorul nu actualizează datele din text field-uri. Rămâne doar să “selectăm” între diferitele acțiuni folosindu-ne de ActionCommand setate în clasa View.

Indiferent de tipul acțiunii, la fiecare nou apel trebuie să creăm noi polinoame (deoarece informațiile se schimbă), dar asta nu înseamnă că modul de creare al acestora diferă. Metoda crearePoli() este prima metodă care face cea dintâi utilizare a string-urilor.

Principala abordare consta in spargerea string-ului. Pentru a nu fi nevoiți să folosim metode puțin diferite în funcție de context, am ales o abordare nu foarte complexa, dar eficientă prin care putem trata polinoame cu coeficienți pozitivi sau negativi.

String monoame[];

String fb,ab;

**if**(s.contains("-")) ab=s.replace("-", "+-");

**else** ab=s;

**if** (ab.charAt(0) == '+') fb=ab.substring(1);

**else** fb= ab;

monoame = fb.split("\\+");

Polinom poli = **new** Polinom(monoame, panel.x);

La final, această metodă trimite la constructorul clasei Polinom un vector de String-uri,

Clasa Polinom

Clasa ce permite crearea efectivă a polinoamelor, ca și colecții de monoame. Ea are un singur constructor public, cel care necesită un vector de String-uri și doi constructori privați ce sunt folosiți strict în structura internă a clasei Polinom, la crearea de polinoame intermediare necesare pentru diferite operații.

La nivelul constructorului public, partea de parsare laborioasă se află în clasa Monom, rezultatul fiind că, în lipsa unei excepții, apelul acestui constructor va crea cu siguranță un obiect de tipul Polinom, ce va conține un ArrayList<Monom> de monoame.

Pe lângă toate acestea, constructorul conține și o metodă de sortare și minificare (*sort\_min()*). În lipsa acestei metode, ne-am putea întâlni cu situații în care am avea, în același polinom, monoame de acelasi grad. Sortarea este mai mult o operație auxiliară, dar benefică, eficientizând alte metode.

**Adunarea a doua polinoame**

În cazul favorabil, adunarea a două polinoame se rezumă la a aduna coeficienții monoamelor, rând pe rând cât timp gradul lor este egal. Totuși, adunarea trebuie să fie valabilă pe caz general.

In cazul adunarii, exista trei cazuri distincte care trebuie tratate. Urmatorul pseudocod trateaza toate aceste situatii, oferind o rezolvare corecta a acestei operatii.



adaugă p2.monom



p1.monom + p2.monom

șterge p2.monom

adaugă ce a rămas

**public** Polinom adunare(Polinom p2) {

Polinom p3 = **new** Polinom(x);

**for**(**int** i=0;i<**this**.monom.size();i++) {

**int** pu1 = **this**.monom.get(i).getPutere();

**int** co1 = **this**.monom.get(i).getCoeficient();

**int** i\_p = p2.IndexPutere(pu1);

**if**(i\_p == -1) {

p3.monom.add(**new** Monom(co1, pu1));

} **else** {

**int** pu3 = p2.monom.get(i\_p).getPutere();

**int** co3 = p2.monom.get(i\_p).getCoeficient();

p3.monom.add(**new** Monom(co1+co3, pu3));

p2.monom.remove(i\_p);

}

}

**for**(**int** j=0;j<p2.monom.size();j++) {

**int** pu2 = p2.monom.get(j).getPutere();

**int** co2 = p2.monom.get(j).getCoeficient();

p3.monom.add(**new** Monom(co2, pu2));

}

**return** p3;

}

**Scaderea a doua polinoame**

Aceasta operatie este tot o adunare, in care coeficientii celui de-al doilea polinom sunt cu semn inversat.

**public** Polinom scadere(Polinom p2) {

Polinom p3;

**for**(**int** i=0;i<p2.monom.size();i++) {

p2.monom.get(i).coeficient = p2.monom.get(i).coeficient\*(-1);

}

p3 = **this**.adunare(p2);

p3.delete();

**return** p3;

}

**Inmultirea a doua polinoame**

**public** Polinom inmultire(Polinom p2) {

Polinom p3 = **new** Polinom(x);

**for**(**int** i=0;i<**this**.monom.size();i++) {

Monom mo1 = **this**.monom.get(i);

**for**(**int** j=0;j<p2.monom.size();j++) {

Monom mo2 = p2.monom.get(j);

Monom mo3 = mo1.inmultire(mo2);

p3.monom.add(mo3);

}

}

p3.sort\_min();

p3.sort\_min();

**return** p3;

}

Aceasta metoda primește ca si parametru un polinom,creand un altul, care reprezinta produsul si pe care il returneaza, nu inainte de a apela sort\_min() pentru a aranja polinomul rezultat.

**Impartirea a doua polinoame**

Aceasta metoda se bazeaza pe scaderea repetata a unui polinom dintr-altul, ceea ce ramane la final fiind restul impartirii.

**public** ArrayList<Polinom> impartire(Polinom impartitor) {

ArrayList<Polinom> p = **new** ArrayList<Polinom>();

Polinom c = **new** Polinom(x);

Polinom r = **new** Polinom(x);

r = **this**;

**if**(r.Grad() < impartitor.Grad()) {

c.monom.add(**new** Monom(0,0));

} **else** {

**while**(r.Grad() >= impartitor.Grad()) {

Monom dmax = cautaMaxim(r);

Monom imax = cautaMaxim(impartitor);

Monom monom = dmax.impartire(imax);

Polinom pmax= **new** Polinom(monom, x);

c.monom.add(monom);

r = r.scadere(impartitor.inmultire(pmax));

}

}

**if**(r.monom.isEmpty()) r.monom.add(**new** Monom(0,0));

p.add(c);

p.add(r);

**return** p;

}

}

Clasa Monom

Clasa care conține operațiile elementare de adunare, scădere, înmulțire și împărțire. Aceste operații sunt relative simple, lucrul cu monoamele fiind mult mai usor in comparative cu polinoamele.

Cea mai complexa metoda din aceasta clasa este metoda de validare a monomului, Ea este o implementare a unui arbore decizional pentru a trata diferitele cazuri ce apar la introducerea datelor.

**if**(!monom.matches("^[a-zA-Z0-9\\^\\\*\\- ]\*")) **return** **false**;

Pattern formatPolinom = Pattern.*compile*("\\^");

Matcher m = formatPolinom.matcher(monom);

String s = **new** String();

**while**(m.find()) {

s = m.group();

}

**if**(s.isEmpty()) {

q = monom.split("[a-zA-Z]");

**if**(q.length == 0) {

coeficient = 1;

putere = 1;

} **else** {

**if**(!q[0].isEmpty())coeficient = Integer.*parseInt*(q[0]);

**else** coeficient = 1;

**if**(q[0] == monom) putere = 0 ;

**else** putere = 1;

}

} **else** {

q = monom.split("\\^");

**try** {

String nString = **new** String();

**for**(**int** i = 0; i < q[0].length(); i++){

**char** a = q[0].charAt(i);

**if**(a == 45) nString = nString + a;

**if**(a > 47 && a < 58) nString = nString + a;

}

**if**(nString.isEmpty()) coeficient = 1 ;

**else** coeficient=Integer.*parseInt*(nString);

putere = Integer.*parseInt*(q[1]);

} **catch**(NumberFormatException e) {

System.***out***.println("Format invalid");

}

}

**return** **true**;

# 5. Rezultate

La o prima privire, toate operatiile implementate functioneaza destul de bine, furnizand un rezultat correct din punct de vedere matematic. In cazul in care polinoamele nu sunt correct scrise de catre utilizator, se va arunca o exceptie, asteptandu-se introducerea unui alt polinom pentru a se putea realiza operatiile implementate.

# 6. Concluzii

Prin această temă consider că am reusit sa aprofundez toate cunostintele de POO învățate semestrul trecut. Am învățat sa folosesc lucruri noi ,precum arhitectura de tip MVC sau folosirea Pattern Matchingului folosind expresii Regex. Consider ca abordarea aleasa este in directia cea buna, dar cu siguranță există multe lucruri care pot fi imbunatatite. O posibila dezvoltare ulterioara este reprezentata de realizarea operatiilor de integrare si derivare, precum si utilizarea Junit pentru testare.

# 7.Bibliografie

<https://dsrl.eu/courses/pt/materials/A1_Support_Presentation.pdf>

<https://dsrl.eu/courses/pt/materials/PT2021-2022_Assignment_1.pdf>

<https://regex101.com/>