DOCUMENTATIE

TEMA 1

CALCULATOR POLINOMIAL

NUME STUDENT: Alexandru Andrei-Denis

GRUPA: 30221

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc95297886)

[3. Proiectare 3](#_Toc95297887)

[4. Implementare 3](#_Toc95297888)

[5. Rezultate 3](#_Toc95297889)

[6. Concluzii si dezvoltari ulterioare 3](#_Toc95297890)

[7. Bibliografie 3](#_Toc95297891)

# Obiectivul temei

* 1. **Obiectivul principal**

Obiectivul principal al temei este reprezentat de crearea unei aplicatii pentru implementarea unui sistem de tip calculator dedicat procesarii polinoamelor(constituiti din mai multi termeni, numiti si monoame) de o singura variabila si cu coeficienti de tip intreg si efectuarea, prin alegere cu ajutorul unei interfete grafice, si vederea rezultatelor unor operatii matematice de baza.

* 1. **Obiective secundare**
* **Dezvoltarea de scenarii si use-case-uri -** In cadrul computerelor, un scenariu reprezinta succesiunea de interactiuni „foreseeable” dintre roluri(cunoscuti in UML - Unified Modeling Language - si sub numele de „*actori*”) si sistemul reprezentat de obicei de catre computer. In cazul acestei teme, rolul este reprezentat de utilizatorul aplicatiei. Fiecare scenariu are un scop(goal). Asemanator, un use-case este o lista de actiuni sau evenimente care definesc un set de interactiuni dintre *actor* si sistem. Mai multe use-case-uri se pot grupa pentru a defini un scenariu. - Capitolul 2
* **Alegerea structurilor de date -** Gasirea unor structuri de date adecvate pentru a duce la capat obiectivul principal - Capitolul 3
* **Impartirea pe clase -** Folosirea unei arhitecturi MVC (Model-View-Controller) pentru a proiecta o interfata grafica (GUI - Graphic User Interface) - Capitolul 3
* **Dezvoltarea algoritmilor -** Capitolul 3
* **Implementarea solutiei -** Descrierea fiecarei clase cu campurile si metodele sale importante, precum si descrierea interfetei pentru utilizator - Capitolul 4
* **Testarea -** Descriera catorva scenarii de testare a operatiilor pe polinoame, folosind ca instrument JUnit Testing

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Utilizarea programului presupune introducerea de catre utilizator (in doua TextField-uri) a celor doua polinoame ce urmeaza a fi folosite pentru efectuarea uneia dintre urmatoarele operatii:

* 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)

Preluarea si efectuarea acesor operatii se va efectua dupa apasarea unor butoane ce au semne specifice pentru operatii, astfel ca intr-un al treilea TextField va fi afisat rezultatul asteptat.

Deoarece e posibil ca unii utilizatori sa introduca polinoame gresite, de fiecare data cand un polinom introdus in TextField are un format gresit fata de cel normal al unui polinom (unul sau mai multe monoame reprezentate sub forma semn(+-, optional), coeficient intreg(optional), variabila(optionala, dar obligatoriu x sau X) urmat de o putere (^a), la randul ei intreaga si optionala), sau sunt introduse diferite Stringuri care nu formeza un polinom (spre exemplu propozitii: “Acesta clar nu este un polinom”), la apasarea unui buton ce declanseaza o operatie, utilizatorul va fi anuntat prin intermediul TextField-ului desemnat rezultatului ca polinomul/polinoamele introdus/e de el nu este/sunt valid/e (nu respecta formatul sau implica o operatie imposibila ex: impartirea la 0). Insa, daca utilizatorul introduce un polinom dezordonat( gradele variabilelor nu sunt in ordine nici crescatoare sau descrescatoare), programul il va lua ca valid, ordonandu-l in timpul procesarii operatiilor. In cazul in care input-urile sunt valide, in cel de-al treilea TextField-ul se va afisa rezultatul.

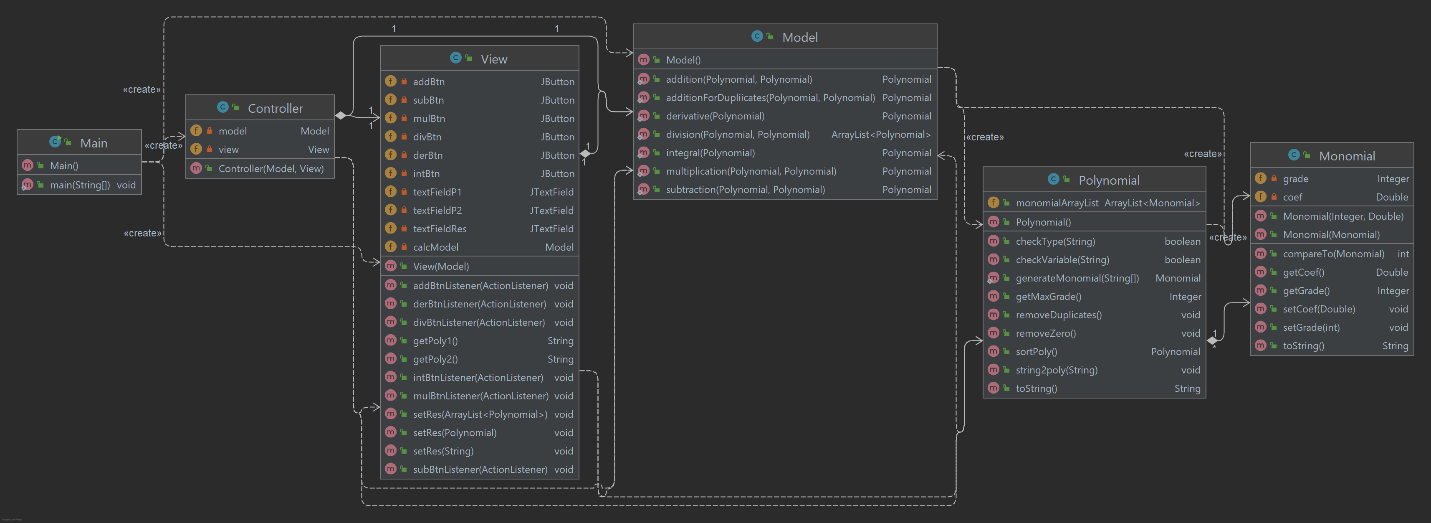
# Proiectare

* 1. **Alegera structurilor de date ­**

Structurile de date principala folosita in dezvoltarea si implementarea aplicatiei au fost monomul, care este definit cu ajutorul clasei Monomial de doua variabile instanta (un intreg pentru reprezentarea gradului si un numar real - Double - pentru reprezentarea coeficientului, ex: 2x^3), si ArrayList-ul de monoame folosit pentru definirea polinomului, ArrayList-ul fiind atat potrivit cat si recomandat in prezentarea suport oferita.

Desi in datele de intrare utilizatorul va fi nevoit sa introduca doar polinoame cu coeficienti intregi, in cazul in care acesta va opta pentru operatia de integrare, pentru reprezentarea corecta a rezultatului va fi nevoie de exprimarea coeficientilor prin intermediul numerelor reale, dar cum si rezultatul este la randul sau un polinom, format din mai multe monoame, am optat pentru o variabila instanta de tip Double pentru reprezentarea coeficientilor.

* 1. **Impartirea pe clase ­­-** Diagrama UML



Pachetele in care au fost imparite clasele prezentate in diagrama UML de mai sus sunt

* Pachetul UI (User Interface) - Clasa View si clasa Controller, responsabile pentru aspectul si functionarea interfetei grafice
* Pachetul Logic - Clasa Monomial si clasa Polynomial, responsabile pentru descierea corecta si eficienta a polinoamelor asupra carora se vor face operatii
* Pachetul Model - Clasa Model si clasa Main. Clasa Model contine metodele necesare pentru implementarea corecta a operatiilor, in timp ce clasa Main contine metoda principala a aplicatiei, care face legatura dintre cele 3 piese de baza ale arhitecturii Model View Controller.
  1. **Algorimti demni de mentionat**

Inmultire - Se parcurg element cu element primul polinom si al doilea polinom (for in for), inmultind fiecare monom din primul cu fiecare monom din al doilea, monomul rezultat in urma unei inmultiri fiind introdus intr-un polinom, care se aduna la rezultatul final. Acest lucru rezolva eficient problema duplicatelor (monoame de acelasi grad prezente in acelasi timp in acelasi polinom).

Impartire - impartirea este relizata in 5 pasi:

* Ordonarea descrescatoare in functie de grad a monoamelor polinoamelor
* Impartirea polinomului de grad mai mare la cel de grad mai mic - in cazul aplicatiei, primul polinom introdus va fi considerat deimpartitul operatiei iar cel de-al doilea polinom va fi impartitorul
* Se imparte primul monom al polinomului P1 la primul monom al polinomului P2 si se obtine primul termen monomial al catului impartirii
* Se inmulteste catul cu polinomul P2 si rezultatul se scade din P1, obtinand astfel restul impartirii
* Repetarea pasilor incepand cu pasul 2, considerand restul ca nou deimpartit al impartirii, pana cand gradul restului va fi mai mic decat gradul impartitorului P2

# Implementare

* 1. **Pachetul Logic**

Clasa **Monomial**

public class Monomial implements Comparable<Monomial>{  
 private Integer grade;  
 private Double coef;

.

. //Constructori, settere si gettere

.

Cum am mentionat anterior, cele doua variabile instanta ale clasei sunt *grade* de tipul intreg si *coef* de tipul Double.

Metode importante:

public String toString() {  
 if (!this.getCoef().equals(0.0)) {  
 if (this.getGrade()==0)  
 {  
 System.*out*.printf("%+.2f ", this.getCoef());  
 return String.*format*("%+.2f ", this.getCoef());  
 }  
 else if(this.getGrade() == 1)  
 {  
 System.*out*.printf("%+.2fx ", this.getCoef());  
 return String.*format*("%+.2fx ", this.getCoef());  
 }  
 else {  
 System.*out*.printf("%+.2fx^%d ",this.getCoef(), this.getGrade());  
 return String.*format*("%+.2fx^%d ",this.getCoef(), this.getGrade());  
 }  
 }  
 return "";  
}

Metoda toString faciliteaza afisarea intr-un mod „friendly” (coeficientul sa fie cu semn si maxim 2 zecimale) si corecta a monomului, tratand special cazurile de grad 0 sau 1.

public int compareTo(Monomial o) {  
 return this.getGrade().compareTo(o.getGrade());  
}

Metoda compareTo, obtinuta cu ajutorul interfetei Comparable, este suprascrisa pentru a realiza comparatia dintre monoame in functie de grad.

Clasa **Polynomial**

public class Polynomial {  
 public ArrayList<Monomial> monomialArrayList = new ArrayList<>();

Singura variabila instanta a acestei clase este ArrayListul de monoame, folosit pentru stocarea monoamelor polinomului intr-o structura ordonata si usor de iterat si accesat.

Metode importante:

public Polynomial sortPoly(){  
 Collections.*sort*(this.monomialArrayList, Collections.*reverseOrder*());  
 return this;  
}

Sorteaza monoamele polinomul in ordine descrescatoare, in functie de gradul acestora

public void removeZero() {  
 Polynomial auxPolynomial = new Polynomial();  
 for (Monomial i : this.monomialArrayList) {  
 if (!i.getCoef().equals(0.0))  
 auxPolynomial.monomialArrayList.add(i);  
 }  
 this.monomialArrayList = auxPolynomial.monomialArrayList;  
}  
  
public void removeDuplicates(){  
 Polynomial auxPolynomial = new Polynomial();  
 for(Monomial i: this.monomialArrayList){  
 Polynomial auxPolynomial2 = new Polynomial();  
 auxPolynomial2.monomialArrayList.add(i);  
 auxPolynomial = Model.*additionForDupliicates*(auxPolynomial2, auxPolynomial);  
 }  
 this.monomialArrayList = auxPolynomial.monomialArrayList;  
}

Cele doua metode sunt responsabile de simplificarea aspectului polinomului, scapandu-l de monoame nule sau de mai multe monoame de acelasi grad.

public static Monomial generateMonomial(String[] stringAux){}

public void string2poly(String polyString){}

Functia string2poly este folosita pentru a converti sirul de caractere primit ca input din interfata grafica intr-un obiect de tip Polynomial. Aceasta sparge sirul in mai multe siruri in functie de gasirea semnului +, urmand mai apoi ca sirurile obtinute sa fie transformate in monoame cu ajutorul functiei generateMonomial.

public boolean checkVariable(String polyString){}

public boolean checkType(String polyString){  
 polyString = polyString.replaceAll("\\s+", "");  
 polyString = polyString.toLowerCase();  
 if(polyString.charAt(0) != '-')  
 polyString = "+" + polyString;  
 String[] stringformatat = polyString.split("[-+][0-9]\*([a-zA-Z](\\^[0-9]\*)?)?\n");  
 for(String s : stringformatat){  
 if(s.isEmpty())  
 return false;  
 }  
 return true;  
}

Functia checkVariable verifica daca variabilele polinomului introdus de utilizator sunt reprezentate ca „x” sau „X”, in timp ce functia checkType verifica daca sirul de caractere ce „se doreste a fi polinom” respecta formatul mentionat anterior: unul sau mai multe monoame reprezentate sub forma semn(+-, optional), coeficient intreg(optional), variabila(optionala, dar obligatoriu x sau X) urmat de o putere (^a), la randul ei intreaga si optionala.

* 1. **Pachetul Model**

Clasa **Model**

Clasa Model nu este inzestrata cu nicio variabila instanta, in schimb contine metodele corespondente operatiilor pe care aplicatia trebuie sa le efectueze. Pe langa cele 6 operatii standard, clasa Model contine si o metoda de adunare folosita in metoda de eliminare a duplicatelor, prezentata anterior

public static Polynomial addition(Polynomial p1, Polynomial p2)

public static Polynomial additionForDupliicates(Polynomial p1, Polynomial p2)

public static Polynomial subtraction(Polynomial p1, Polynomial p2)

public static Polynomial multiplication(Polynomial p1, Polynomial p2)

public static ArrayList<Polynomial> division(Polynomial p1, Polynomial p2)

public static Polynomial derivative(Polynomial p1)

public static Polynomial integral(Polynomial p1)

In cadrul integrarii, polinomul se va afisa fara constanta, chiar daca integrala este una nedefinita. In cazul integrarii lui 0, in TextField-ul de rezultat se va afisa „C”.

* 1. **Pachetul UI**

Clasa **View**

public class View extends JFrame

In clasa View sunt instantiate cele 3 TextField-uri pentru polinoame, precum si 6 cele butoane corespunzatoare celor 6 operatii. Pe langa acestea se adauga si variabila instanta de tip Model a Vederii, necesara pentru functionarea corecta a aplicatiei sub arhitectura Model View Controller.

Metodele din clasa View constau in preluarea polinoamelor sub forma de String din TextField-uri, trimiterea unui polinom sub forma de String catre al 3-lea TextField a rezultatului, precum si in adaugarea de Listenere celor 6 butoane

public String getPoly1(){  
 return textFieldP1.getText();  
}

public void addBtnListener(ActionListener AddBtnListener){  
 addBtn.addActionListener(AddBtnListener);  
}

public void setRes(String res){  
 textFieldRes.setText(res);  
}  
//res pentru rezultat  
public void setRes(Polynomial res){  
 textFieldRes.setText(res.toString());  
}  
//res pentru impartire  
public void setRes(ArrayList<Polynomial> arrayList){  
 String str = "Q: " + arrayList.get(0).toString() +" R: "+ arrayList.get(1).toString();  
 textFieldRes.setText(str);  
}

Metoda setRes, este supraincarcata deoarece in TextField-ul rezultatului suntem nevoiti sa afisam un polinom, mai multe polinoame (in cazul impartirii avem cat-Q si rest-R) sau un mesaj de eroare in cazul introducerii unor date incorecte de catre utilizator.

Clasa **Controller**

Clasa este responsabila cu crearea unei legaturi intre Model si View fara ca acestea sa „stie” una de alta. In interiorul acesteia exista subclase pentru definirea Listenerelor pentru fiecare buton din interfata grafica.

public class Controller {  
 private Model model;  
 private View view;  
  
 public Controller(Model model, View view){  
 this.model = model;  
 this.view = view;  
  
 view.addBtnListener(new AddBtnListener());  
 view.subBtnListener(new SubBtnListener());  
 view.mulBtnListener(new MulBtnListener());  
 view.divBtnListener(new DivBtnListener());  
 view.intBtnListener(new IntBtnListener());  
 view.derBtnListener(new DerBtnListener());  
 }

Exemplu de Listener:

class IntBtnListener implements ActionListener{  
 public void actionPerformed(ActionEvent e){  
 String polynomial1 = view.getPoly1();  
 if(polynomial1.isEmpty())  
 view.setRes("More input needed for the first polynomial");  
 else {  
 Polynomial p1 = new Polynomial();  
 p1.string2poly(polynomial1);  
 if (p1.toString().charAt(0) == '0' && p1.toString().length() == 1) {  
 view.setRes("C");  
 } else {  
 Polynomial res = Model.*integral*(p1);  
 view.setRes(res);  
 }  
 }  
 }  
}

* 1. **Clasa Main**

public class Main {  
 public static void main(String[] args) {  
  
 Model model = new Model();  
 View view = new View(model);  
 Controller controller = new Controller(model, view);  
 view.setVisible(true);}

}

# Rezultate

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

In cadrul testarii cu JUnit, am luat cate 5 teste pentru fiecare operatie

* Adunare
  + (x^3) + (x^2) = x^3 + x^2
  + (x^2 + x + 1) + (-2x) = x^2 - x + 1 --- cazul in care un polinom incepe cu coeficient negativ
  + (4x^3 - 3x^2 + 2) + 6x = 4x^3 - 3x^2 + 6x + 2 --- monom de grad ce nu se regaseste in polinomul 1
  + (x^3 + x + 7) + 0 = x^3 + x + 7 --- adunare cu polinom nul
  + (-2x^3 + 6x^3) + (x^2) = 4x^3 + x^2 --- mai multe monoame de grad egal in acelasi polinom
* Scadere
  + (x^3) - (x^2) = x^3 - x^2
  + (x + x^2) - (x^3) = -x^3 + x^2 + x --- grade mai mare al polinoamului scazut
  + (x^3) - (-3x^3 + 2x - 5) = (4x^3 - 2x + 5) --- scadere de tip - cu - fac +
  + (x^3 + 1) - (-x^2 -5 + x) = x^3 + x^2 - x + 6 --- termeni aranjati in alta ordine
  + 2 - (x^2 + 4x - 5x^4 + 3x + 2) =5x^4 - x^2 - 7x --- termeni aranjati in alta ordine, mai multe monoame de grad egal
* Inmultire
  + (x^3 + 1) (-x^2 -5 + x) = -5 + x - x^2 - 5x^3 + x^4 - x^5 --- grad mai mare inmultit cu grad mai mic, negativ
  + (x^3 + 1) \* 0 = 0 --- inmultirea cu polinomul nul
  + (x^3 + 1) (-3x^2 + 5x -4) = -3x^5 + 5x^4 - 4x^3 - 3x^2 + 5x - 4 --- inmultirea cu polinom cu mai multe monoame
  + (x^3 + x^3 + 2) (x^2 - x^2 + 1) = 2x^3 + 2 --- inmultirea cu 1 si reducerea monoamelor duplicate inainte de efectuarea operatiei de inmultire
  + (-1 -3x + 4x^2)(-4 +5x +5) = 20 x^3 - 11 x^2 - 8 x - 1 --- asemenea cu inmultirea de mai sus
* Impartire
  + (x^3 - 2x^2 + 6x -5) / (x^2 -1) = Q: x-2 R: 7x - 7 ---chiar impartirea din exemplu, cu cat si rest nenule
  + (x^2 + 2x + 1) / (x+1) = Q: x+1 R: 0 --- formula de calcul prescurtat, impartire cu rest 0
  + (x^3 + 1) / (x + 1) = Q: x^2 - x + 1 R: 0 --- formula de calcul prescurtat pentru gradul 3, impartire cu rest 0
  + (x^2 + 1) / (x^3) = Q: 0 R: x^2 + 1 --- impartirea unui polinom de grad mai mic la un polinom de grad mai mare, catul va fi 0 iar restul va fi chiar deimpartitul
* Derivare
  + (3x^2 + 2x + 1)’ = 6x + 2 --- derivare polinom de grad 2
  + (3x^4 + 2x + 1)’ = 12x^3 + 2 --- derivare polinom de grad 4
  + (x^3 - 2x^2 + 6x -5)’ = 3x^2 - 4x + 6 --- derivare polinom de grad 3 cu semne alternante ale monoamelor
  + 4’ = 0 --- derivarea de constante
  + (9x)’ = 9 --- derivarea unui polinom de grad 1
* Integrare
  + ∫ (3x^2 + 2x + 1) = x^3 + x^2 + x
  + ∫ (5x^4 + 2x + 1) = x^5 + x^2 + x
  + ∫ 4 = 4x
  + ∫ 10x = 5x^2

# Concluzii si dezvoltari ulterioare

In urma analizarii codului si testarii aplicatiei, personal concluzionez ca aceasta este una total functionala atata timp cat datele de intrare respecta conventia definita mai sus. Pe langa testele facute cu JUnit, am efectual o multitudine de teste direct in aplicatie, folosind atat cod, cat si interfata grafica (cateva probabil pot fi gasite in codul sursa, comentate, deoarece nu le-am observat inainte de incarcare), si consider ca functioneaza corect in toate subcazurile la care am reusit sa ma gandesc.

Ce am invatat din aceasta tema? In primul rand m-am familiarizat cu crearea unei interfete grafice doar din cod, anterior folosindu-ma de WindowBuildere de tip drag and drop, cum ar fi cel din NetBeans. De asemenea, alt mecanism cu care nu am mai lucrat pana acum si pentru care am avut nevoie de o aprofundare a fost testarea unitara cu JUnit, aceasta tema fiind prima data cand m-am folosit de acest aspect.

In cazul dezvoltarii ulterioare, aplicatia ar putea benificia de o interfata grafica mai bogata din punct de vedere al aspectului. Din punct de vedere logic, pentru ca o integrare sa fie absolut corecta, ar trebui sa fie afisata si constanta C. Acest lucru ar putea fi facilitat de adaugarea clasei Polynomial a unei variabile instanta de tip boolean care ar putea semnala daca polinomul este integrat sau nu, iar in caz adevarat, acest atribut sa comande functia toString sa adauge la afisare un „+C”. O alta dezvoltare ar putea fi gasirea unui algoritm mai eficient de a elimina monoamele duplicate dintr-un polinom. In cazul de fata se parcurge polinomul monom cu monom, monomul se plaseaza intr-un polinom auxiliar, care dupa fiecare plasare se aduna cu polinomul rezultat. Problema aparuta este ca metoda removeDuplicates are nevoie de operatia de adunare, in timp ce pentru a oferi un rezultat corect, metoda addition apeleaza removeDuplicates. Pentru a rezolva aceasta problema, am definit o noua metoda de adunare, identica cu aceasta, numita additionForDuplicates, care nu apeleaza removeDuplicates la finalul acesteia. Solutia este functionala, insa personal nu consider ca este una eleganta.

# Bibliografie

<https://softwareengineering.stackexchange.com/questions/347498/difference-between-scenario-and-use-case>

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

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

<https://www.youtube.com/watch?v=i_Qn-d6WExs&>

<https://towardsdatascience.com/regular-expressions-clearly-explained-with-examples-822d76b037b4>

<https://stackoverflow.com/questions/35821071/learning-guis-setcontentpane-method>

<https://www.javaprogramto.com/2020/04/java-collection-sort-custom-sorting.html>