**CALCULATOR DE POLINOAME**

Tema 1

DOCUMENTAȚIE

**Gavrilescu Andreea-Lavinia**

**Grupa 30228**

# CUPRINS

[1. **Obiectivul temei** 3](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297885)

[2. **Analiza problemei, modelare, scenarii, cazuri de utilizare** 3](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297886)

[3. **Proiectare** 3](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297887)

[4. **Implementare** 3](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297888)

[5. **Rezultate** 3](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297889)

[6. **Concluzii** 3](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297890)

[7. **Bibliografie** 3](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297891)

Obiectivul temei

**Obiective principale**

* Proiectarea și implementarea unui calculator de polinoame cu o interfață garfică

interactivă prin care utilizatorii pot introduce polinoame, selecta operația dorită și să obțină rezultatul.

**Obiective secundare**

* Analizarea problemei și identificarea cerințelor
* Proiectarea calculatorului de polinoame
* Implementarea calculatorului de polinoame
* Testarea calculatorului de polinoame

Analiza problemei, modelare, scenarii, cazuri de utilizare

**Analiza**

Calculator de polinoame

**Cerințe funcționale**

* Calculatorul de polinoame ar trebui să permită introducerea a doua polinoame
* Calculatorul de polinoame ar trebui să permită utilizatorilor să selecteze operații matematice
* Calculatorul de polinoame ar trebui să efectueze adunarea a doua polinoame
* Calculatorul de polinoame ar trebui să efectueze scăderea a doua polinoame
* Calculatorul de polinoame ar trebui să efectueze înmulțirea a doua polinoame
* Calculatorul de polinoame ar trebui să efectueze împărțirea a doua polinoame
* Calculatorul de polinoame ar trebui să efectueze derivarea unui polinom
* Calculatorul de polinoame ar trebui să efectueze integrarea unui polinom

**Cerințe non-funcționale**

* Calculatorul de polinoame ar trebui să fie interactiv
* Calculatorul de polinoame ar trebui să fie ușor de folosit de către utilizator

**Cazuri de utilizare**

* Caz de utilizare 1:adunarea a doua polinoame

Actor principal: utilizatorul

Scenariu principal

1. Utilizatorul introduce 2 polinoame în interfața grafică
2. Utilizatorul selectează operația de adunare apăsând butonul corespunzător
3. Calculatorul de polinoame efectuează operația de adunare și afișează rezulatul

Scenariu alternativ

1. Utilizatorul introduce polinoame incorecte (ex. : cu una sau mai multe variabile)
2. Scenariul se întoarce la punctul 1.

* Caz de utilizare 2:scăderea a doua polinoame

Actor principal: utilizatorul

Scenariu principal

1. Utilizatorul introduce 2 polinoame în interfața grafică

2. Utilizatorul selectează operația de scădere apăsând butonul corespunzător

3. Calculatorul de polinoame efectuează operația de scădere și afișează rezulatul

Scenariu alternativ

1. Utilizatorul introduce polinoame incorecte (ex. : cu una sau mai multe variabile)

2. Scenariul se întoarce la punctul 1.

* Caz de utilizare 3:înmulțirea a doua polinoame

Actor principal: utilizatorul

Scenariu principal

1. Utilizatorul introduce 2 polinoame în interfața grafică

2. Utilizatorul selectează operația de înmulțire apăsând butonul corespunzător

3. Calculatorul de polinoame efectuează operația de înmulțire și afișează rezulatul

Scenariu alternativ

1. Utilizatorul introduce polinoame incorecte (ex. : cu una sau mai multe variabile)

2. Scenariul se întoarce la punctul 1.

* Caz de utilizare 4:împărțirea a doua polinoame

Actor principal: utilizatorul

Scenariu principal

1. Utilizatorul introduce 2 polinoame în interfața grafică

2. Utilizatorul selectează operația de împărțire apăsând butonul corespunzător

3. Calculatorul de polinoame efectuează operația de împărțire și afișează rezulatul

Scenariu alternativ

1. Utilizatorul introduce polinoame incorecte

2. Scenariul se întoarce la punctul 1.

* Caz de utilizare 5:derivarea a doua polinoame

Actor principal: utilizatorul

Scenariu principal

1. Utilizatorul introduce un polinom în interfața grafică

2. Utilizatorul selectează operația de derivare apăsând butonul corespunzător

3. Calculatorul de polinoame efectuează operația de derivare și afișează rezulatul

Scenariu alternativ

1. Utilizatorul introduce polinoame incorecte

2. Scenariul se întoarce la punctul 1.

* Caz de utilizare 6:integrarea a doua polinoame

Actor principal: utilizatorul

Scenariu principal

1. Utilizatorul introduce un polinom în interfața grafică

2. Utilizatorul selectează operația de integrare apăsând butonul corespunzător

3. Calculatorul de polinoame efectuează operația de integrare și afișează rezulatul

Scenariu alternativ

1. Utilizatorul introduce polinoame incorecte

2. Scenariul se întoarce la punctul 1.

Proiectare

**Design general**

**x^3 + 2x^2 + 3**

Calculator de polinoame

**REZULTAT**

**x^4 + 2x^3 – 8x^2**

**OPERAȚIE**

**Divizare în pachete**

Conține clasele care implementează interfața

Interfața grafică

Modele

Logica

Conține clasele care implementează operațiile

Conține clasele care modelează datele aplicației

Pentru afișarea rezulatelor potrivite confrom datelor introduse de utilizatori prin interfața grafică voi folosi un Model-View-Controller pattern care împarte aplicația în 3 categorii: procesare, input și output.

* Componentele Model – încapsulează datele și funcționalitatea
* Componentele View – afișează informația utilizatorului – obține datele afișate din model
* Controller – fiecare view are un controller asociat; acesta primește un input, de obicei un eveniment care denotă mișcarea mouse-ului, apăsarea mouse-ului sau a tastaturii; aceste evenimente sunt traduse ca apeluri care sunt trimise către model sau view.

Controller-ul modifică apeluri View-ului în funcție de modificările din Model

Datele din Model sunt folosite în View

Model

View-ul poate cere date din Model

View

Controller

Interacțiunea cu View-ul face apeluri către Controller

**Divizare în clase**

Monom

Model

Polinom

Operații

CalculatorController

Controller

View

ViewController

A screenshot of a computer

Description automatically generated with medium confidence

Pachetul Models - conține 3 clase pentru a asigura lizibilitatea codului, dar și intuitivitatea, astfel:

* Clasa Monom – conține 2 atribute numite sugestiv, coeficient și exponent, precum și 2 constructori (cu parametrii diferiți), precum și metodele de get(), toString() și equals(Object other).
* Clasa Polinom – are ca și atribut o lista de monoame, o metodă standard de get(), getMonomMaxim(), sortareDupaExponent(), toString(), equals(Object other).
* Clasa Operații – conține metodele statice de implementare a operațiilor: adunare, scădere, înmulțire, împărțire, derivare și integrare.

Pachetul View – conține clasa CalculatorView care implementează interfața folosind Java Swing.

Pachetul Controller – conține clasa CalculatorController care receptează acțiunile utilizatorului și returnează rezultatul potrivit prin interfața grafică.

Clasa Main – care are doua atribute, view-ul și controller-ul.

Implementare

1. **Clasa Monom** – are ca atribute private, coeficientul și exponentul, de tip double, respectiv int.

Avem 2 constructori: primul care primește coeficientul și exponentul, iar al doilea primește expresia String a unui monom pe care o desparte cu ajutorul unui pattern Regex în coeficient și exponent.

// Constructorul clasei Monom cu parametrii corespunzatori  
public Monom*(*double coeficient, int exponent*) {* this.coeficient = coeficient;  
 this.exponent = exponent;  
*}*public Monom*(*String expresie*) {* //System.out.println(expresie);  
  
 // Daca expresia monomului dat nu contine caracterele ^ si x, inseamna ca exponentul lui x este 0.  
 if *(*!expresie.contains*(*"^"*)* && !expresie.contains*(*"x"*)) {* this.coeficient = Integer.*parseInt(*expresie*)*; // covertim valoarea de tipul String a coeficientului in int  
 this.exponent = 0;  
 *}* // Altfel, exponentul este mai mare sau egal cu 1.  
 else *{* // \\\*? = sa contina caracterul \* de 0 sau 1 dati  
 // x = sa contina caracterul x  
 // \\^? = sa contina caracterul ^ de 0 sau 1 dati  
  
 // O lista in care retinem pattern-ul pentru despartire  
 List*<*String*>* list = List.*of(*expresie.split*(*"\\\*?x\\^?"*))*;  
  
 String coeficientString = list.get*(*0*)*; // retinem coeficientul de tipul String  
  
 // Daca coeficientul de tipul String are semnul -  
 if *(*coeficientString.equals*(*"-"*))* this.coeficient = -1; // punem valoarea -1 in coeficientul de tip int  
  
 // Daca coeficientul de tip String are semnul + sau este gol (nu are coeficient)  
 else if *(*coeficientString.equals*(*"+"*)* || coeficientString.isEmpty*())* this.coeficient = 1; // punem valoarea -1 in coeficientul de tip int  
  
 // Altfel, convertim valoarea coeficientul de tip String la tipul int  
 else  
 this.coeficient = Integer.*parseInt(*coeficientString*)*;  
  
 // Daca lista obtinuta in urma split-ului are mai mult de un element, inseamna ca exponentul este mai mare  
 // decat 1.  
 if *(*list.size*()* != 1*) {* String exponentString = list.get*(*1*)*; // retinem valoarea de tipul String a exponentului  
 this.exponent = Integer.*parseInt(*exponentString*)*; // apoi o convertim la tipul int  
 *}* // Altfel, inseamna ca exponentul este 1.  
 else  
 this.exponent = 1;  
 *}  
}*

Dupa care urmează metodele de get() pentru coeficient și exponent.

Iar apoi, în metoda toString vom afișa monomul în funcție de exponent, coeficient, cât și semnul acestuia.

// Metoda pentru afisare monoamelor unui polinom  
@Override  
public String toString*() {* // Daca coeficientul este 1  
 if *(*this.coeficient == 1*) {* // Daca exponentul este 1 (x^1 = x)  
 if *(*this.exponent == 1*)* return "x"; // afisam doar 'x'  
  
 // Daca exponentul este 0  
 if *(*this.exponent == 0*)* return String.*valueOf(*this.coeficient*)*; // afisam doar valoarea coeficientului (x^0 = 1)  
  
 // Altfel, se afiseaza "x^"  
 else  
 return "x^" + this.exponent;  
 *}* // Altfel, daca coeficientul este -1  
 else if *(*this.coeficient == -1*) {* // Daca exponentul este 1 (x^1 = x)  
 if *(*this.exponent == 1*)* return "-x";  
  
 // Daca exponentul este 0  
 if *(*this.exponent == 0*)* return String.*valueOf(*this.coeficient*)*; // afisam valoarea coeficentului (x^0 = 1)  
  
 // Altfel afisam "-x^"  
 else  
 return "-x^" + this.exponent;  
 *}* else if *(*this.coeficient == 0*) {* // Daca exponentul este 1 (x^1 = x)  
 if *(*this.exponent >= 1*)* return "";  
  
 // Daca exponentul este 0  
 else  
 return String.*valueOf(*this.coeficient*)*; // afisam valoarea coeficentului (x^0 = 1)  
 *}* // Altfel, daca coeficientul nu e 1 sau -1  
 else *{* // Daca exponentul este 1 (x^1 = x)  
 if *(*this.exponent == 1*)* return this.coeficient + "\*x"; // afisam "coefiecient\*x"  
  
 // Daca exponenetul este 0 (x^0 = 1)  
 if *(*this.exponent == 0*)* return String.*valueOf(*this.coeficient*)*; // aifsam doar coeficientul  
  
 // Altfel, daca coeficeintul nu e 1 sau -1  
 else  
 return this.coeficient + "\*x^" + this.exponent; // afisam "coeficient\*x^exponent"  
 *}  
}*

Metoda equals() ne ajută să verifică dacă doua monoame sunt egale.

@Override  
public boolean equals*(*Object other*) {* if *(*!*(*other instanceof Monom*))* return false;  
  
 Monom ot = *(*Monom*)* other;  
  
 return this.exponent == ot.getExponent*()* && this.coeficient == ot.getCoeficient*()*;  
*}*

2. **Clasa Polinom** – are ca atribut private o lisă de monoame, de tipul Monom, precum și doi constructori: unul care primește ca parametru o lista de monoame, iar celălalt primește o expresie de tipul String pe care o va separa în monoame folosind un pattern Regex.

public Polinom*(*List*<*Monom*>* monoame*) {* this.monoame = new ArrayList*<>(*monoame*)*;  
*}*// Constructor care primeste ca parametru o expresie de tipul "x^2+2\*x+1"  
public Polinom*(*String expresie*) {* // Intializam lista de monoame  
 this.monoame = new ArrayList*<>()*;  
  
 //int i = 0;  
  
 // Pentru a desparti expresia polinomului se folosteste regex  
 // \\\*? = sa contina caracterul \* de 0 sau 1 dati  
 // x = sa contina caracterul x  
 // \\^? = sa contina caracterul ^ de 0 sau 1 dati  
 Pattern spl = Pattern.*compile(*"*([*+-*]*?*[*^-+*]*+*)*"*)*;  
 Matcher potrivire = spl.matcher*(*expresie*)*;  
  
 while *(*potrivire.find*()) {* //i++;  
 Monom monom = new Monom*(*potrivire.group*(*1*))*;  
 this.monoame.add*(*monom*)*;  
 //System.out.println("Monom " + i + ": " + monom);  
 *}  
}*

Alte doua metode importante sunt:

// Metoda pentru determinarea monomului de grad maxim  
public Monom getMonomMaxim*() {* List*<*Monom*>* cpy = new ArrayList*<>(*this.monoame*)*;  
  
 cpy.sort*(*Comparator.*comparing(*Monom::getExponent, Comparator.*reverseOrder()))*;  
  
 return cpy.get*(*0*)*;  
*}*// Sortare inversă după exponent  
public void sortareDupaExponent*() {* this.monoame.sort*(*Comparator.*comparing(*Monom::getExponent, Comparator.*reverseOrder()))*;  
*}*

// Afisarea polinomului folosind lista de monoame  
@Override  
public String toString*() {* // Initialiam un String gol  
 String str = "";  
  
 // Parcurgem lista de monoame (for each)  
 for *(*Monom m : monoame*) {* if *(*m.getCoeficient*()* > 0*)* // daca coeficientul e pozitiv  
 str += "+"; // punem '+'  
  
 str += m; // apoi adaugam monomul  
 *}* return str; // afisam String-ul corespunzator monomului  
*}*@Override  
public boolean equals*(*Object other*) {* if *(*!*(*other instanceof Polinom*))* return false;  
  
 Polinom ot = *(*Polinom*)* other;  
 this.sortareDupaExponent*()*;  
 ot.sortareDupaExponent*()*;  
  
 if *(*ot.getMonoame*()*.size*()* != this.monoame.size*())* return false;  
  
 for *(*int i = 0; i < ot.getMonoame*()*.size*()*; i++*)* if *(*!ot.getMonoame*()*.get*(*i*)*.equals*(*this.monoame.get*(*i*)))* return false;  
  
 return true;  
*}*

3. **Clasa Operații** – conține metodele statice de implementare a operațiilor matematice de: adunare, scădere, înmulțire și împărțire. Toate sunt implementate după același princpiu: stocarea monoamelor din lista de monoame într-un hash map pentru ușurarea și eficientizarea realizării operațiilor. Vom folosi un hash map rezultat în care (în cazul operațiilor de adunare si scadere) îl vom inițializa cu valoarea primului polinom și vom aduna și scadea direct valoarea celui de-al doilea polinom, pe când la celelalte vom pune rezultatul în timpul efectuării operațiilor.

// Metode pentru operatiile corespunzatoare butoanelor  
public static Polinom adunarePolinoame*(*Polinom polinom1, Polinom polinom2*) {* // Declaram 2 liste de monoame, una corespunzatoare polinomului introdus, iar cealalta polinomului rezultat  
 List*<*Monom*>* monoame = new ArrayList*<>()*;  
 List*<*Monom*>* rezultat = new ArrayList*<>()*;  
  
 // Declaram 2 HaspMap-uri pentru cele doua polinoame introduse  
 Map*<*Integer, Double*>* map1 = new HashMap*<>()*;  
 Map*<*Integer, Double*>* map2 = new HashMap*<>()*;  
  
 // Parcurgem lista de monoame a primului polinom (for each)  
 for *(*Monom m : polinom1.getMonoame*())* map1.put*(*m.getExponent*()*, m.getCoeficient*())*; // punem in cheie - exponentul, iar in valoare - coeficientul  
  
 // Procedam la fel si pentru polinomul al doilea  
 for *(*Monom m : polinom2.getMonoame*())* map2.put*(*m.getExponent*()*, m.getCoeficient*())*;  
  
 // In HashMapu-ul corespunzator rezultatului, punem HashMap-ul corespunzator primului polinom  
 Map*<*Integer, Double*>* mapRezultat = new HashMap*<>(*map1*)*;  
  
 // Parcurgem cheile (adica exponentii)  
 for *(*Integer key : map2.keySet*()) {* // Daca map-ul rezultat (adica implicit map1) contine cheia din map2  
 if *(*mapRezultat.containsKey*(*key*)) {* // In map-ul rezultat, punem peste valoare, adunarea cu valoarea map-ului 2  
 mapRezultat.put*(*key, mapRezultat.get*(*key*)* + map2.get*(*key*))*;  
  
 // Daca o valoare este 0  
 if *(*mapRezultat.get*(*key*)* == 0*)* mapRezultat.remove*(*key*)*; // o eliminam  
 *}* else  
 // Altfel, daca nu contine cheia, valoarea respectiva din map-ul rezultat  
 // primeste valoarea corespunzatoare din map2  
 mapRezultat.put*(*key, map2.get*(*key*))*;  
  
 *}* // Parcurgem cheile din map-ul rezultat  
 for *(*Integer key : mapRezultat.keySet*()) {* double value = mapRezultat.get*(*key*)*; // retinem valoarea  
  
 monoame.add*(*new Monom*(*value, key*))*; // adugam in lista de monoame cheile si valorile,  
 // adica coeficientii si exponentii  
  
  
 *}* // Daca nu avem niciun monom ramas la final, inseamna ca rezultatul este 0,  
 // deci returnam un polinom nou cu expresia "0".  
 if *(*monoame.size*()* == 0*)* return new Polinom*(*"0"*)*;  
  
 // returnam polinomul creat din lista de monoame obtinuta mai sus  
 Polinom rez = new Polinom*(*monoame*)*;  
 rez.sortareDupaExponent*()*;  
 return rez;  
*}*public static Polinom scaderePolinoame*(*Polinom polinom1, Polinom polinom2*) {* // Declaram 2 liste de monoame, una corespunzatoare polinomului introdus, iar cealalta polinomului rezultat  
 List*<*Monom*>* monoame = new ArrayList*<>()*;  
 List*<*Monom*>* rezultat = new ArrayList*<>()*;  
  
 // Declaram 2 HaspMap-uri pentru cele doua polinoame introduse  
 Map*<*Integer, Double*>* map1 = new HashMap*<>()*;  
 Map*<*Integer, Double*>* map2 = new HashMap*<>()*;  
  
 // Parcurgem lista de monoame a primului polinom (for each)  
 for *(*Monom m : polinom1.getMonoame*())* map1.put*(*m.getExponent*()*, m.getCoeficient*())*; // punem in cheie - exponentul, iar in valoare - coeficientul  
  
 // Procedam la fel si pentru polinomul al doilea  
 for *(*Monom m : polinom2.getMonoame*())* map2.put*(*m.getExponent*()*, m.getCoeficient*())*;  
  
 // In HashMapu-ul corespunzator rezultatului, punem HashMap-ul corespunzator primului polinom  
 Map*<*Integer, Double*>* mapRezultat = new HashMap*<>(*map1*)*;  
  
 // Parcurgem cheile (adica exponentii)  
 for *(*Integer key : map2.keySet*()) {* // Daca map-ul rezultat (adica implicit map1) contine cheia din map2  
 if *(*mapRezultat.containsKey*(*key*)) {* // In map-ul rezultat, punem peste valoare, scaderea cu valoarea map-ului 2  
 mapRezultat.put*(*key, mapRezultat.get*(*key*)* - map2.get*(*key*))*;  
  
 // Daca o valoare este 0  
 if *(*mapRezultat.get*(*key*)* == 0*)* mapRezultat.remove*(*key*)*; // o eliminam  
 *}* else  
 // Altfel, daca nu contine cheia, valoarea respectiva din map-ul rezultat  
 // primeste valoarea corespunzatoare din map2  
 mapRezultat.put*(*key, map2.get*(*key*))*;  
  
 *}* // Parcurgem cheile din map-ul rezultat  
 for *(*Integer key : mapRezultat.keySet*()) {* double value = mapRezultat.get*(*key*)*; // retinem valoarea  
  
 monoame.add*(*new Monom*(*value, key*))*; // adugam in lista de monoame cheile si valorile,  
 // adica coeficientii si exponentii  
 *}* // Daca nu avem niciun monom ramas la final, inseamna ca rezultatul este 0,  
 // deci returnam un polinom nou cu expresia "0".  
 if *(*monoame.size*()* == 0*)* return new Polinom*(*"0"*)*;  
  
 // returnam polinomul creat din lista de monoame obtinuta mai sus  
 Polinom rez = new Polinom*(*monoame*)*;  
 rez.sortareDupaExponent*()*;  
 return rez;  
  
*}*public static Polinom inmultirePolinoame*(*Polinom polinom1, Polinom polinom2*) {* if *((*polinom1.getMonoame*()*.size*()* == 1 && polinom1.getMonoame*()*.get*(*0*)*.getCoeficient*()* == 0*)* ||  
 polinom2.getMonoame*()*.size*()* == 1 && polinom2.getMonoame*()*.get*(*0*)*.getCoeficient*()* == 0*)* return new Polinom*(*"0"*)*;  
  
 // Declaram o lista de monoame  
 List*<*Monom*>* monoame = new ArrayList*<>()*;  
  
 // Declaram 3 HaspMap-uri pentru cele doua polinoame introduse si una pentru rezultat  
 Map*<*Integer, Double*>* map1 = new HashMap*<>()*;  
 Map*<*Integer, Double*>* map2 = new HashMap*<>()*;  
 Map*<*Integer, Double*>* mapRezultat = new HashMap*<>()*;  
  
 // Parcurgem lista de monoame a primului polinom (for each)  
 for *(*Monom m : polinom1.getMonoame*())* map1.put*(*m.getExponent*()*, m.getCoeficient*())*; // punem in cheie - exponentul, iar in valoare - coeficientul  
  
 // Procedam la fel si pentru polinomul al doilea  
 for *(*Monom m : polinom2.getMonoame*())* map2.put*(*m.getExponent*()*, m.getCoeficient*())*;  
  
 // Parcurgem cheile (adica exponentii)  
 for *(*Integer key1 : map1.keySet*()) {* for *(*Integer key2 : map2.keySet*()) {* int exp = key1 + key2;  
 double coef = map1.get*(*key1*)* \* map2.get*(*key2*)*;  
  
 if *(*!mapRezultat.containsKey*(*exp*))* mapRezultat.put*(*exp, coef*)*;  
  
 else  
 mapRezultat.put*(*exp, coef + mapRezultat.get*(*exp*))*;  
 *}  
 }* // Parcurgem cheile din map-ul rezultat  
 for *(*Integer key : mapRezultat.keySet*()) {* double value = mapRezultat.get*(*key*)*; // retinem valoarea  
  
 monoame.add*(*new Monom*(*value, key*))*; // adugam in lista de monoame cheile si valorile,  
 // adica coeficientii si exponentii  
 *}* // returnam polinomul creat din lista de monoame obtinuta mai sus  
 Polinom rez = new Polinom*(*monoame*)*;  
 rez.sortareDupaExponent*()*;  
 return rez;  
*}*public static Monom impartireMonoame*(*Monom monom1, Monom monom2*) {* // Declaram și intializam atributele polinomului rezultat dupa impartirea a doua monoame  
 double coeficient = monom1.getCoeficient*()* / monom2.getCoeficient*()*;  
 int exponent = monom1.getExponent*()* - monom2.getExponent*()*;  
  
 // returnam monomul rezultat  
 return new Monom*(*coeficient, exponent*)*;  
*}*public static List*<*Polinom*>* impartirePolinoame*(*Polinom deimpartit, Polinom impartitor*){* // Declararea a doua liste, una a monoamelor polinoameor primite si una pentru polinomul rezultat dupa impartire  
 List*<*Polinom*>* result = new ArrayList*<>()*;  
 List*<*Monom*>* monoame = new ArrayList*<>()*;  
  
 // Daca exponentul monomului maxim al deimpartitului este mai mare  
 // decat cel al impartitorui, operatia se poate realiza  
 while *(*deimpartit.getMonomMaxim*()*.getExponent*()* >= impartitor.getMonomMaxim*()*.getExponent*()) {* // Stocam monoamele maxime, dupa care efectuam operatia de impartire a acestora  
 Monom md = deimpartit.getMonomMaxim*()*;  
 Monom mi = impartitor.getMonomMaxim*()*;  
 Monom mc = Operatii.*impartireMonoame(*md, mi*)*;  
  
 // Adaugam monomul rezultat la lista de monoame  
 monoame.add*(*mc*)*;  
  
 // Continuam sa efectuam operatia conform schemei lui Horner, actualizand variabilele conform acesteia  
 Polinom aux = Operatii.*inmultirePolinoame(*impartitor, new Polinom*(*List.*of(*mc*)))*;  
 deimpartit = Operatii.*scaderePolinoame(*deimpartit, aux*)*;  
 *}* // Catul primeste valoarea listei de monoame transformata în polinom  
 Polinom cat = new Polinom*(*monoame*)*;  
  
 // Sortam descrescator catul si restul(stocat in variabila deimpartit) dupa exponent  
 cat.sortareDupaExponent*()*;  
 deimpartit.sortareDupaExponent*()*;  
  
 // Adaugam catul si restul in lista de polinoame, dupa care returnam rezultatul  
 result.add*(*cat*)*;  
 result.add*(*deimpartit*)*;  
  
 return result;  
*}*public static Polinom derivarePolinoame*(*Polinom polinom*) {* // Declaram o lista de monoame  
 List*<*Monom*>* monoame = new ArrayList*<>()*;  
  
 // Declaram un HaspMap pentru polinomul introdus  
 Map*<*Integer, Double*>* map = new HashMap*<>()*;  
  
 // Parcurgem lista de monoame a polinomului (for each)  
 for *(*Monom m : polinom.getMonoame*())* if *(*m.getExponent*()* != 0*)* map.put*(*m.getExponent*()*, m.getCoeficient*())*; // punem in cheie - exponentul, iar in valoare - coeficientul  
  
 // Declaram HashMap-ul corespunzator rezultatului  
 Map*<*Integer, Double*>* mapRezultat = new HashMap*<>()*;  
  
 // Parcurgem cheile (adica exponentii)  
 for *(*Integer key : map.keySet*()) {* mapRezultat.put*(*key - 1, key \* map.get*(*key*))*;  
 *}* // Parcurgem cheile din map-ul rezultat  
 for *(*Integer key : mapRezultat.keySet*()) {* double value = mapRezultat.get*(*key*)*; // retinem valoarea  
  
 monoame.add*(*new Monom*(*value, key*))*; // adugam in lista de monoame cheile si valorile,  
 // adica coeficientii si exponentii  
 *}* // Daca nu avem niciun monom ramas la final, inseamna ca rezultatul este 0,  
 // deci returnam un polinom nou cu expresia "0".  
 if *(*monoame.size*()* == 0*)* return new Polinom*(*"0"*)*;  
  
 // returnam polinomul creat din lista de monoame obtinuta mai sus  
 Polinom rez = new Polinom*(*monoame*)*;  
 rez.sortareDupaExponent*()*;  
 return rez;  
  
*}*public static Polinom integrarePolinoame*(*Polinom polinom*) {* // Declaram o lista de monoame  
 List*<*Monom*>* monoame = new ArrayList*<>()*;  
  
 // Declaram un HashMap corespunzator polinomului rezultat  
 Map*<*Integer, Double*>* mapRezultat = new HashMap*<>()*;  
  
 // Declaram 2 HaspMap-uri pentru cele doua polinoame introduse  
 Map*<*Integer, Double*>* map = new HashMap*<>()*;  
  
 // Parcurgem lista de monoame a polinomului (for each)  
 for *(*Monom m : polinom.getMonoame*())* map.put*(*m.getExponent*()*, m.getCoeficient*())*; // punem in cheie - exponentul, iar in valoare - coeficientul  
  
 // Parcurgem cheile (adica exponentii)  
 for *(*Integer key : map.keySet*()) {* mapRezultat.put*(*key + 1, map.get*(*key*)* / *(*key + 1*))*;  
 *}* // Parcurgem cheile din map-ul rezultat  
 for *(*Integer key : mapRezultat.keySet*()) {* double value = mapRezultat.get*(*key*)*; // retinem valoarea  
  
 monoame.add*(*new Monom*(*value, key*))*; // adugam in lista de monoame cheile si valorile,  
 // adica coeficientii si exponentii  
 *}* // returnam polinomul creat din lista de monoame obtinuta mai sus  
 Polinom rez = new Polinom*(*monoame*)*;  
 rez.sortareDupaExponent*()*;  
 return rez;  
  
*}*

4. **Clasa View** – impementează elementele interfeței grafice

– conține atributele necesare celor 4 casete text, a etichetelor corepunzătoare acestora, precum și a butoanelor operațiilor matematice precizate mai sus.

În constructor vom poziționa toate elementele interfeței.

// Constructorul clasei CalculatorView  
public CalculatorView*() {* // Marginile ferestrei interfetei  
 this.setBounds*(*100, 100, 608, 648*)*;  
 this.setDefaultCloseOperation*(*JFrame.*EXIT\_ON\_CLOSE)*;  
 this.getContentPane*()*.setLayout*(*null*)*;  
  
 // Etichetele din interfata  
 titlteLabel = new JLabel*(*"Calculator de polinoame"*)*;  
 titlteLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 22*))*;  
 titlteLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 titlteLabel.setBounds*(*105, 10, 405, 49*)*;  
 this.getContentPane*()*.add*(*titlteLabel*)*;  
  
 firstLabel = new JLabel*(*"Primul polinom"*)*;  
 firstLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 firstLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 firstLabel.setBounds*(*34, 98, 161, 51*)*;  
 this.getContentPane*()*.add*(*firstLabel*)*;  
  
 secondLabel = new JLabel*(*"Al doilea polinom"*)*;  
 secondLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 secondLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 secondLabel.setBounds*(*34, 189, 172, 51*)*;  
 this.getContentPane*()*.add*(*secondLabel*)*;  
  
 rezultatLabel = new JLabel*(*"Rezultat"*)*;  
 rezultatLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 rezultatLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 rezultatLabel.setBounds*(*34, 282, 172, 49*)*;  
 this.getContentPane*()*.add*(*rezultatLabel*)*;  
  
 restLabel = new JLabel*(*"Rest"*)*;  
 restLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 restLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 restLabel.setBounds*(*34, 362, 172, 51*)*;  
 this.getContentPane*()*.add*(*restLabel*)*;  
  
 // Casetele text in care se vor introduce cele doua polinoame, respectiv unde se va afisa rezultatul  
 firstTextField = new JTextField*()*;  
 firstTextField.setBounds*(*230, 101, 266, 49*)*;  
 this.getContentPane*()*.add*(*firstTextField*)*;  
 firstTextField.setColumns*(*10*)*;  
  
 secondTextField = new JTextField*()*;  
 secondTextField.setColumns*(*10*)*;  
 secondTextField.setBounds*(*230, 189, 266, 49*)*;  
 this.getContentPane*()*.add*(*secondTextField*)*;  
  
 rezultatTextField = new JTextField*()*;  
 rezultatTextField.setColumns*(*10*)*;  
 rezultatTextField.setBounds*(*230, 282, 266, 49*)*;  
 this.getContentPane*()*.add*(*rezultatTextField*)*;  
  
 restTextField = new JTextField*()*;  
 restTextField.setColumns*(*10*)*;  
 restTextField.setBounds*(*230, 362, 266, 49*)*;  
 this.getContentPane*()*.add*(*restTextField*)*;  
  
 // Butoanele corespunzatoare operatiilor de pe interfata  
 adunareButton = new JButton*(*"Adunare"*)*;  
 adunareButton.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 adunareButton.setBounds*(*48, 472, 172, 49*)*;  
 this.getContentPane*()*.add*(*adunareButton*)*;  
  
 scadereButton = new JButton*(*"Scadere"*)*;  
 scadereButton.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 scadereButton.setBounds*(*48, 538, 172, 49*)*;  
 this.getContentPane*()*.add*(*scadereButton*)*;  
  
 inmultireButton = new JButton*(*"Inmultire"*)*;  
 inmultireButton.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 inmultireButton.setBounds*(*230, 472, 172, 49*)*;  
 this.getContentPane*()*.add*(*inmultireButton*)*;  
  
 impartireButton = new JButton*(*"Impartire"*)*;  
 impartireButton.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 impartireButton.setBounds*(*230, 538, 172, 49*)*;  
 this.getContentPane*()*.add*(*impartireButton*)*;  
  
 derivareButton = new JButton*(*"Derivare"*)*;  
 derivareButton.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 derivareButton.setBounds*(*412, 472, 172, 49*)*;  
 this.getContentPane*()*.add*(*derivareButton*)*;  
  
 integrareButton = new JButton*(*"Integrare"*)*;  
 integrareButton.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 20*))*;  
 integrareButton.setBounds*(*412, 538, 172, 49*)*;  
 this.getContentPane*()*.add*(*integrareButton*)*;  
  
 // Afisare fereastra actuala  
 this.setVisible*(*true*)*;  
*}*

Pe lângă metodele de get() pentru castele text, avem metode pentru a verifica dacă unul dintre butoane este apăsat, caz în care se trimite un apel către controller pentru a afișa rezulatul corepunzător.

Exemplu:

public void addAdunareListener*(*ActionListener actionListener*) {* this.adunareButton.addActionListener*(*actionListener*)*;  
*}*

O altă metodă este cea care este apelată în caz că introducem un input greșit.

// Metoda pentru input necorespunzator  
public void showMessage*(*String message*) {* JOptionPane.*showMessageDialog(*this, message*)*;  
*}*

5. **Clasa Controller** – primește un apel când este apăsat un buton de pe interfață, iar după efectuarea operației corespunzătoare butonului, returnează rezultatul în caseta specifică.

Constructorul primește ca parametru view-ul, iar acesta are rolul de a lega butoanele de metoda care apelează rezolvarea operației corespunzătoare.

public CalculatorController*(*CalculatorView calculatorView*) {* this.calculatorView = calculatorView;  
  
 // Punem listeners pe butoane pentru a executa operatia corespunzatoare la apasare  
 this.calculatorView.addAdunareListener*(*new AdunareListener*())*;  
 this.calculatorView.addScadereListener*(*new ScadereListener*())*;  
 this.calculatorView.addInmultireListener*(*new InmultireListener*())*;  
 this.calculatorView.addImpartireListener*(*new ImpartireListener*())*;  
 this.calculatorView.addDerivareListener*(*new DerivareListener*())*;  
 this.calculatorView.addIntegrareListener*(*new IntegrareListener*())*;  
  
*}*

Apoi avem metodele care implementeaza interfața ActionListener prin care se verifică inputul primit după apăsarea unui buton.

Exemple:

class AdunareListener implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*) {* try *{* // Luam valorile din casetele text de pe interfata  
 String polString1 = calculatorView.getFirstTextField*()*;  
 String polString2 = calculatorView.getSecondTextField*()*;  
  
 if *(*polString1.equals*(*""*))* throw new Exception*(*"Nu s-a introdus nimic in prima caseta!"*)*;  
  
 if *(*polString2.equals*(*""*))* throw new Exception*(*"Nu s-a introdus nimic in a doua caseta!"*)*;  
  
 // Intializam doua polinoame care primesc datele de pe interfata  
 Polinom polinom1 = new Polinom*(*polString1*)*;  
 Polinom polinom2 = new Polinom*(*polString2*)*;  
  
 // In caseta rezultat, afisam polinomul rezultat  
 calculatorView.getRezultatTextField*()*.setText*(*Operatii.*adunarePolinoame(*polinom1, polinom2*)*.toString*())*;  
 calculatorView.getRestTextField*()*.setText*(*""*)*;  
  
 *}* catch *(*NumberFormatException exception*) {* calculatorView.showMessage*(*"Polinom invalid!"*)*;  
 *}* catch *(*Exception exception*) {* // Prindem exceptiile pentru input gresit  
 calculatorView.showMessage*(*exception.getMessage*())*;  
 //exception.printStackTrace();  
 *}  
 }  
}*

class ImpartireListener implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*) {* try *{* // Luam valorile din casetele text de pe interfata  
 String polString1 = calculatorView.getFirstTextField*()*;  
 String polString2 = calculatorView.getSecondTextField*()*;  
  
 // Intializam doua polinoame care primesc datele de pe interfata  
 Polinom polinom1 = new Polinom*(*polString1*)*;  
 Polinom polinom2 = new Polinom*(*polString2*)*;  
  
 // Stocam rezultatul impartirii(cat si rest) intr-o lista de polinoame  
 List*<*Polinom*>* list1 = Operatii.*impartirePolinoame(*polinom1, polinom2*)*;  
  
 // In caseta rezultat, afisam catul impartirii, iar in caseta rest, restul  
 calculatorView.getRezultatTextField*()*.setText*(*list1.get*(*0*)*.toString*())*;  
 calculatorView.getRestTextField*()*.setText*(*list1.get*(*1*)*.toString*())*;  
  
 *}* catch *(*Exception exception*) {* // Prindem exceptiile pentru input gresit  
 calculatorView.showMessage*(*"Bad Input"*)*;  
 //exception.printStackTrace();  
 *}  
   
 }  
}*

class DerivareListener implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*) {* try *{* // Luam valorile din casetele text de pe interfata  
 String polString1 = calculatorView.getFirstTextField*()*;  
 String polString2 = calculatorView.getSecondTextField*()*;  
  
 if *(*polString1.equals*(*""*)) {* // Intializam polinomul care primeste datele de pe interfata  
 Polinom polinom2 = new Polinom*(*polString2*)*;  
  
 // In caseta rezultat, afisam polinomul rezultat  
 calculatorView.getRezultatTextField*()*.setText*(*Operatii.*derivarePolinoame(*polinom2*)*.toString*())*;  
 *}* else *{* // Intializam polinomul care primeste datele de pe interfata  
 Polinom polinom1 = new Polinom*(*polString1*)*;  
  
 // In caseta rezultat, afisam polinomul rezultat  
 calculatorView.getRezultatTextField*()*.setText*(*Operatii.*derivarePolinoame(*polinom1*)*.toString*())*;  
 *}* if *(*polString2.equals*(*""*)) {* // Intializam polinomul care primeste datele de pe interfata  
 Polinom polinom1 = new Polinom*(*polString1*)*;  
  
 // In caseta rezultat, afisam polinomul rezultat  
 calculatorView.getRezultatTextField*()*.setText*(*Operatii.*derivarePolinoame(*polinom1*)*.toString*())*;  
 *}* else *{* // Intializam polinomul care primeste datele de pe interfata  
 Polinom polinom2 = new Polinom*(*polString2*)*;  
  
 // In caseta rezultat, afisam polinomul rezultat  
 calculatorView.getRezultatTextField*()*.setText*(*Operatii.*derivarePolinoame(*polinom2*)*.toString*())*;  
 *}  
  
 }* catch *(*Exception exception*) {* // Prindem exceptiile pentru input gresit  
 calculatorView.showMessage*(*"Bad Input"*)*;  
 //exception.printStackTrace();  
 *}  
 }  
}*

6. **Clasa Main** – are atributele view și controller-ul care primește ca parametru view-ul respectiv, precum si metoda main.

public static void main*(*String*[]* args*) {* // Initilizam view-ul  
 CalculatorView view = new CalculatorView*()*;  
  
 // Initializam controller-ul, al carui constructor primeste ca parametru view-ul  
 CalculatorController controller = new CalculatorController*(*view*)*;  
*}*

Rezultate

Pentru testarea proiectului s-a folosit Junit în vederea testării unui număr distinct de polinoame pentru a descperi eventuale erori, însă toate testele au rulat cu succes.

Vor apărea erori dacă se va introduce un input greșit sau dacă nu se completează caseta text corespunzătoare polinoamelor necesare efectuării unei operații (exemplu implementat doar pentru adunare).

Testarea cu Junit se face în clasa TestMain.

Exemplu de test:

public void testAdaugare*() {* Polinom polinom1 = new Polinom*(*"5\*x^2"*)*;  
 Polinom polinom2 = new Polinom*(*"6\*x^2+5"*)*;  
 Polinom polinom3 = new Polinom*(*"2\*x^4+x^3+3\*x^2"*)*;  
 Polinom polinom4 = new Polinom*(*"5\*x^3+2\*x"*)*;  
 Polinom polinom5 = new Polinom*(*"x^6+2\*x"*)*;  
 Polinom polinom6 = new Polinom*(*"-x^6+4\*x^3"*)*;  
 Polinom polinom7 = new Polinom*(*"x^5"*)*;  
 Polinom polinom8 = new Polinom*(*"-x^5"*)*;  
 Polinom polinom9 = new Polinom*(*"4\*x^3-2\*x"*)*;  
 Polinom polinom10 = new Polinom*(*"-2\*x^3"*)*;  
  
 Polinom rezultat1 = new Polinom*(*"11\*x^2+5"*)*;  
 Polinom rezultat2 = new Polinom*(*"2\*x^4+6\*x^3+3\*x^2+2\*x"*)*;  
 Polinom rezultat3 = new Polinom*(*"4\*x^3+2\*x"*)*;  
 Polinom rezultat4 = new Polinom*(*"0"*)*;  
 Polinom rezultat5 = new Polinom*(*"2\*x^3-2\*x"*)*;  
 Polinom rezultat6 = new Polinom*(*"-x^6+9\*x^3+2\*x"*)*;  
 Polinom rezultat7 = new Polinom*(*"x^5+5\*x^2"*)*;  
 Polinom rezultat8 = new Polinom*(*"9\*x^3"*)*;  
 Polinom rezultat9 = new Polinom*(*"x^6-2\*x^3+2\*x"*)*;  
 Polinom rezultat10 = new Polinom*(*"-x^5+5\*x^2"*)*;  
  
 *assertEquals(*rezultat1, Operatii.*adunarePolinoame(*polinom1, polinom2*))*;  
 *assertEquals(*rezultat2, Operatii.*adunarePolinoame(*polinom3, polinom4*))*;  
 *assertEquals(*rezultat3, Operatii.*adunarePolinoame(*polinom5, polinom6*))*;  
 *assertEquals(*rezultat4, Operatii.*adunarePolinoame(*polinom7, polinom8*))*;  
 *assertEquals(*rezultat5, Operatii.*adunarePolinoame(*polinom9, polinom10*))*;  
 *assertEquals(*rezultat6, Operatii.*adunarePolinoame(*polinom6, polinom4*))*;  
 *assertEquals(*rezultat7, Operatii.*adunarePolinoame(*polinom7, polinom1*))*;  
 *assertEquals(*rezultat8, Operatii.*adunarePolinoame(*polinom4, polinom9*))*;  
 *assertEquals(*rezultat9, Operatii.*adunarePolinoame(*polinom5, polinom10*))*;  
 *assertEquals(*rezultat10, Operatii.*adunarePolinoame(*polinom1, polinom8*))*;  
  
*}*

Concluzii

Acest proiect mi-a consolidat cunoștințele teoretice și practice de Java pe care le-am învățat în semestrul 1 prin volumul de lucru mult mai intens și avansat pe care l-am depus în scopul finalizării acestei teme.

Ca dezvoltare ulterioară, se mai poate îmbunătăți aspectul interfeței ca să arate mai interesant, se mai poate adăuga o tastatură pentru a facilita introducerea polinoamelor, putând fi utlizată de elevii care învață pentru prima dată aceste noțiuni, cât și de cei care au nevoie mai departe de aceste cunoștințe pe parcursul facultății.

Bibliografie

* <https://dsrl.eu/courses/pt/materials/A1_Support_Presentation.pdf>
* <https://www.w3schools.com/java/java_regex.asp>
* <https://regex101.com/>
* <https://www.w3schools.com/java/java_hashmap.asp>
* <https://tugofweb.com/2021/01/02/polinoame-impartirea-polinoamelor-teorema-impartirii-cu-rest-teorema-restului-schema-lui-horner/>
* <https://www.emathhelp.net/en/calculators/algebra-1/polynomial-calculator/>