

**Facultatea de Automatică și Calculatoare**

Calculator de Polinoame

Documentație

Pop Crina-Maria

Grupa 30228, An 2

Cuprins

1. Obiectivul temei
2. Analiza problemei
3. Proiectare
   1. Diagrame UML
4. Implementare
   1. Clase
   2. Metode
   3. GUI
5. Rezultate
6. Concluzii
7. Bibliografie

**1.Obiectiv**

Obiectivul principal al primei teme de laborator a fost să gândim, să proiectăm și să implementăm un calculator de polinoame de o singura variabilă, având coeficienți întregi. Acesta calculator de polinoame trebuie să realizeze diferite operații între 2 polinoame (adunare, scădere, înmulțire, împărțire, derivare și integrare). În afara părții de implementare propriu-zisă a acestui sistem de calcul, se poate folosi și o interfață grafică ”prientenoasă” cu utilizatorul, pentru a fi ușor de utilizat de către orice persoană care are cunoștințe minime de matematică .

Subobiectivele sunt reprezentate de analiza problemei și identificarea cerințelor, proiectarea calculatorului, implementarea și testarea pentru a verifica funcșionarea corectă a tuturor operațiilor .

**2. Analiza problemei**

Ce este de fapt un polinom? Un polinom este o expresie construită din termeni numiți monoame, care la rândul lor sunt alcătuite dintr-o constantă (numită coeficient), înmulțită cu una sau mai multe variabile (în cazul nostru o singură variabilă x). Variabila poate avea un exponent constant întreg pozitiv .

Reazlizarea calculatorului se poate face în multe forme, totul depinzând de viziunea programatorului: modul în care se pot introduce polinoamele, modul în care sunt așezate monoamele (daca sunt sortate sau nu), aspectul interfeței grafice etc.

În diagrama de mai jos se pot observa use-case-urile pentru sistemul meu de calcul, iar mai jos este descris pentru exemplificare unul dintre ele.

Calculator polinomial

Alege operația

Aduna

Scade

polinoamele

Inmulteste

Derivează

Integreaza

Rezultat

Utilizator

Use-Case: adunarea polinoamelor

Actor principal: utilizatorul

Scenariul principal corect de funcționare:

1. Utilizatorul introduce cele doua polinoame în interfața grafică.
2. Utilizatorul selectează butonul ”add”
3. Calculatorul de polinoame face adunarea celor două polinoame și afișează rezultatul.

Scenariul alternativ de funcționare: Polinoame incorecte

Utilizatorul introduce polinoame incorecte (cu 2 sau mai multe variabile)

Scenariul se intoarce la pasul 1.

Din această diagramă se pot extrage câteva cerințe funcționale și non-funcționale ale sistemului nostru de calcul.

Cerințele funcționale ale problemei sunt:

* Calculatorul polinomial ar trebui să permită utilizatorului să introducă două polinoame
* Calculatorul polinomial ar trebui să permită utilizatorului să aleagă operația matematică dorită.
* Calculatorul polinomial ar trebui să adune două polinoame
* Calculatorul polinomial ar trebui să scadă două polinoame
* Calculatorul polinomial ar trebui să înmulțească două polinoame
* Calculatorul polinomial ar trebui să împartă două polinoame
* Calculatorul polinomial ar trebui să deriveze un polinom
* Calculatorul polinomial ar trebui să integreze un polinom

Cerințele non-funcționale sunt reprezentate de următoarele:

* Calculatorul polinomial ar trebui să fie ușor de folosit de către utilizator.
* Calculatorul polinomial ar trebui să lucreze rapid, fără ca utilizatorul să fi nevoit să aștepte prea mult timp rezultatul.
* Calculatorul polinomial ar trebui să fie compatibil pe mai multe sisteme de operare, pentru a putea fi utilizat de cât mai multă lume.

**3.Proiectare**

Schema bloc a sistemului este una foarte simplă, pentru că avem trei intrări, și anume cele două polinoame și selectarea operației dorite și ieșirea care ne va da rezultatul în urma operației.

**Calculator de polinoame**

2x^2+3x^1+1

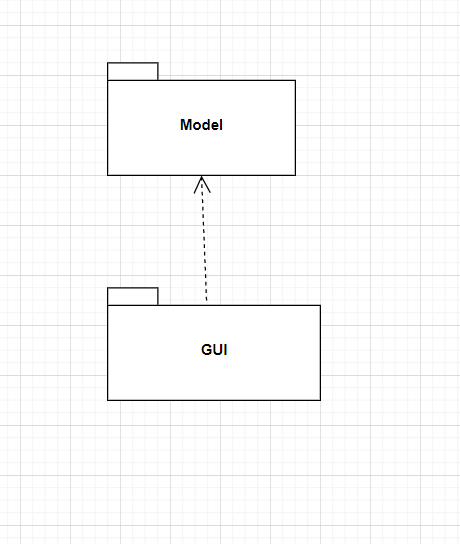
5x^2+2x^1

Operație

Rezultat

**3.1. Diagrame UML**

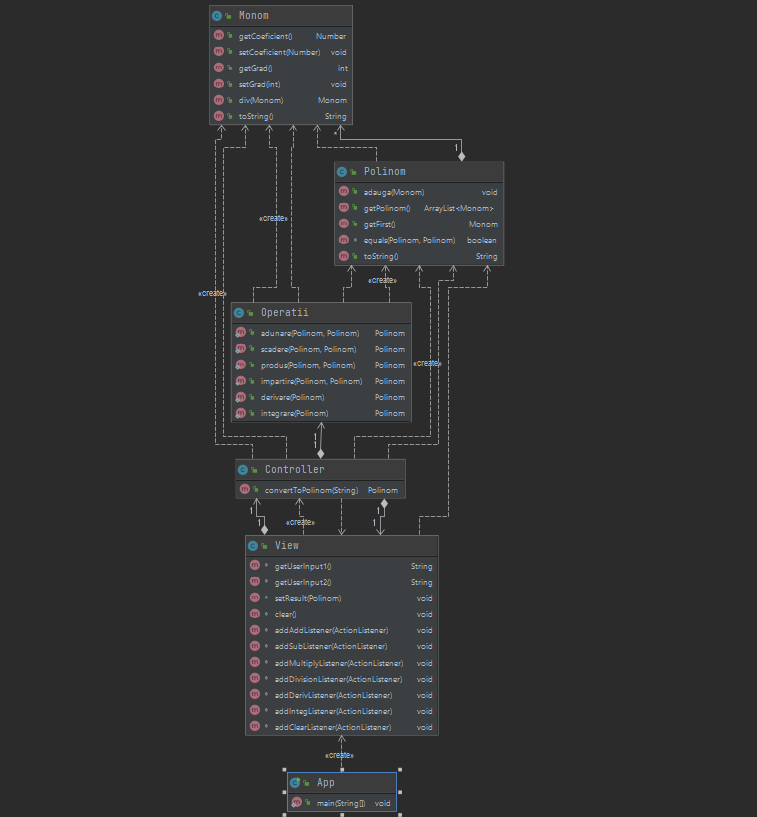
După ce ne-am gândit la o vedere de ansamblu a sistemului nostru de calcul, putem să începem să împărțim problema în sub-probleme, pe care dacă le combinăm ne vor duce la soluția finală.

Am inceput prin a mă gândi la pachetele de care am nevoie. Eu mi-am creat două, GUI și Model. Diagrama de pachete se află mai jos.

Prin aceste două pachete m-am gândit să implementez calculatorul prin pattern-ul arhitectural Model View Controller. Acest pattern îmi împarte logica aplicației în 3: o parte de interfață grafică cu care interacționează utilizatorul, o parte de gestiune a evenimentelor generate de acțiunile utilizatorului și o parte pentru funcționalitățile calculatorului.

1. Modelul reprezintă intern datele.
2. View-ul este folosit pentru rezprezentarea vizuală a datelor.
3. Controller-ul care preia intrarea de la utilizator și o transpune în schimbări în model.

După această etapă, am făcut împărțirea pe clase. Diagrama UML de clase, cu toate legăturile dintre ele este următoarea:



Se poate observa că legăturile au fost făcute de jos în sus.

**4.Implementare**

**4.1. Clase**

Pentru aplicația mea, am ales să implementez 6 clase. Două dintre ele se află în pachetul GUI (Controller și View), celelalte 3 sunt în pachetul Model (Polinom, Monom, Operații), iar clasa App este situată în afara acestora.

**Clasa Monom –** această clasă modelează forma unui monom, având ca variabile instanță un coeficient de tip Number și un exponent întreg (sau un grad).

**Clasa Polinom –** în această clasă mi-am format polinomul din mai multe monoame, astfel variabila instanțî a acestei clase este un ArrayList de monoame.

**Clasa Operatii** – în această clasă am ales să implementez toate operațiile pe care calculatorul nostru polinomial le poate face.

**Clasa View** – aici mi-am creat interfața grafică, adăugând campurile și butoanele neccesare pentru ca utilizatorul să poată folosi aplicația.

**Clasa Controller** – această clasă face cumva legătura dintre programul în sine și interfața utilizator. Prin ea se intercepteazaă evenimentele generate de utilizator pentru a putea să ii ofere la final un rezultat.

**Clasa App** – este cea în care se află metoda main de unde se lansează funcționarea calculatorului, adica ne deschide fereastra aplicației.

**4.2. Metode**

**Metodele din clasa Monom**

Dupa cum am mai amintit, clasa monom are ca variabile instanță un coeficient și un exponent care sunt private. Pentru a putea crea un obiect de tip monom avem un constructor cu 2 parametrii ca să setăm valorile variabilelor, iar pentru a le accesa am creat metodele getCoeficient(), setCoeficient(), getGrad() și setGrad().

public Number getCoeficient() {  
 return coeficient;  
}  
  
public void setCoeficient(Number coef) {  
 coeficient = coef;  
}

Celelalte două sunt identice.

Pe langă acestea, mai există o metodă pentru a împărți două monoame, metodă utilă la împarțirea polinoamelor (public Monom div(Monom m1)), insă din păcate nu am reușit să fac împărțirea să fucționeze. Mtoda toString() am implementat-o pentru a putea afișa într-un mod plăcut polinomul ca un sir de monoame.

**Metodele din clasa Polinom**

În clasa Polinom avem un ArrayList de Monoame ca variabilă instanța pe care ni-l creăm odata cu aplearea constructorului. Pentru a nu rămâne NULL și a putea lucra cu el am implementat metoda adauga() care ne va adaua pe rând câte un monom.

public void adauga(Monom m) {  
 monom.add(m);  
}

De asemenea, mai avem o metoda care va face posibila folosirea în alte clase a ArrayListului denumit monom care este privat, și anume getPolinom().

public ArrayList<Monom> getPolinom() {  
 return monom;  
}

Am mai implementat o metodă getFirst() care să returneze primul element din polinom pentru a fi util la împărțire. La fel ca în clasa Monom, avem și aici metodă toString() care practic parcurge șirul de monoame și le afișsează pe rând.

**Metodele din clasa Operatii**

În această clasă avem toate metodele pentru operațiile pe care le poate face calculatorul de polinoame.

**Adunarea:** pentru operația de adunare am implementat metoda public static Polinom adunare(Polinom p1, Polinom p2). După cum se vede are doua polinoame ca parametru. Ideea de la operația de adunarea am luat-o dintr-un pseudocod găsit pe intrnet și am încercat să îl fac sa meargă după cum scria acolo, dar nu mergea și astfel am modificat puțin.

1. grad p1.monom < oricare grad p2.monom

adauga p2.monom

1. grad p1.monom = oricare grad p2.monom

adauga p1.monom + p2.monom

* 1. daca mai exista grad p2.monom

adauga ce a ramas

Ceea ce am făcut eu în implementarea metodei a fost să verific dacă gradul primul polinom este mai mic decat gradul celui de-al doilea polinom atunci adaug al doilea polinom la rezultat, altfel daca sunt egale fac suma coeficientilor și daca este mai mare gradul primului polinom l-am adaugat pe el la rezultat. La final am mai parcurs polinoamele din locul în care am ramas sa adaug la rezultat ceea ce a mai ramas.

**Scăderea:** la scădere am pornit de la ideea că această operație este de fapt o adunare cu semn schimbat. Astfel, am înmulțit coeficienții celui de-al doilea polinom cu -1 și apoi am apelat metoda de adunare. La sfârșit am resetat coeficienții polinomului al doilea la valoarea lor dinainte.

public static Polinom scadere(Polinom p1, Polinom p2) {  
 Polinom dif = new Polinom();  
  
 for(Monom mon2: p2.getPolinom()) {  
 int coef = (int) mon2.getCoeficient();  
 coef = coef \* -1;  
 mon2.setCoeficient(coef);  
 }  
  
 dif = *adunare*(p1,p2);  
  
 for(Monom mon2: p2.getPolinom()) {  
 int coef = (int) mon2.getCoeficient();  
 coef = coef \* -1;  
 mon2.setCoeficient(coef);  
 }  
  
 return dif;  
}

**Înmulțire:** la înmulțire am parcurs cu două for-uri cele două polinoame și am înmulțit efectiv fiecare termen cu fiecare termen.

public static Polinom produs(Polinom p1,Polinom p2) {  
 Polinom prod = new Polinom();  
 ArrayList<Monom> m1 = p1.getPolinom();  
 ArrayList<Monom> m2 = p2.getPolinom();  
 int gr1,gr2, c1,c2;  
 for(Monom mon1: m1) {  
 for(Monom mon2: m2) {  
 gr1 = mon1.getGrad();  
 gr2 = mon2.getGrad();  
 c1= (int) mon1.getCoeficient();  
 c2 = (int) mon2.getCoeficient();  
 prod.adauga(new Monom(c1 \* c2, gr1 + gr2));  
 }  
 }  
 return prod;  
}

**Împărțire**: această operație nu funcționează, insă am implementat-o după pseudocodul clasic găsit pe internet.

**function** n / d **is**

require d ≠ 0

q ← 0

r ← n // At each step n = d × q + r

**while** r ≠ 0 **and** degree(r) ≥ degree(d) **do**

t ← lead(r) / lead(d) // Divide the leading terms

q ← q + t

r ← r − t × d

**return** (q, r)

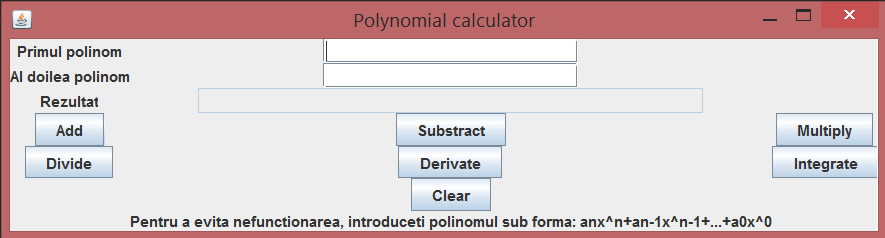
**Derivare:** pentru a putea deriva un polinom am urmat regula pe care o cunoasțtem de la matematică și anume că se înmulțește coeficientul cu exponentul, iar apoi se scade gradul monomului respectiv.

public static Polinom derivare(Polinom p) {  
 Polinom deriv = new Polinom();  
 ArrayList<Monom> m1 = p.getPolinom();  
 for(int i=0; i < m1.size(); i++) {  
 int gr = m1.get(i).getGrad();  
 int coef = (int)m1.get(i).getCoeficient() \* gr;  
 deriv.adauga(new Monom(coef,gr - 1));  
 }  
 return deriv;  
}

**Integrare:** asemănător cu ideea de la derivare, doar ca acestă operație este inversa celeilalte, am împărțit coeficientul cu gradul monomului + 1, iar apoi am crescut gradul cu 1. Aici a trebuit să fiu atentă la împărțirea cu 0, de aceea am un if suplimentar.

public static Polinom integrare(Polinom p) {  
 Polinom integ = new Polinom();  
 ArrayList<Monom> m1 = p.getPolinom();  
 double coef;  
 for(int i=0; i < m1.size(); i++) {  
 int grad = m1.get(i).getGrad();  
 coef = (int) m1.get(i).getCoeficient();  
 if(grad != 0) {  
 integ.adauga(new Monom(coef\*1.0 / (grad + 1),grad + 1));  
 } else {  
 integ.adauga(new Monom(coef, grad + 1));  
 }  
 }  
 return integ;  
}

**4.3. GUI**

****

Interfața grafică sau Graphical User Interface este o interfață cu utilizatorul bazată pe un sistem de afișaj ce utilizează elemente grafice.

Pentru a-mi construi interfața cu utilizatorul am pornit de la câteva idei pe care le-am folosit în semestrul I, schimbând doar unele lucruri. Nu este o interfață foarte complicată, stilizată, pentru că am mers de la început pe ideea de funcționare corectă, lăsând la o parte design-ul. Prin acest fapt, eu consider că este o interfață personalizată, care ar mai putea suferi totușiîmbunătățiri. M-am folosit de componentele din pachetul JavaSwing pentru a o crea.

Astfel interfața mea conține următoarele elemente:

**Frame-ul**: rama în care se adaugă toate componentele din View, este baza interfeței grafice. Acest frame l-am instanțiat în main-ul aplicației mele, de unde va porni execuția, iar tot acolo am denumit această fereastră și am setat opțiunea ca la apăsaarea butonului ”X” ea să se închidă.

**Panel-ul**: acesta are rolul de a conține toate componentele necesare pentru ca utilizatorul sa poata realiza operațiile dorite.

**Butoane:** interfața mea conține 7 butoane, cu diferite funcționalități:

* Add – folosit la adunarea celor doua polinoame
* Substract - folosit la scaderea celor doua polinoame
* Multiply - folosit la înmulțirea celor doua polinoame
* Divide - folosit la împărțirea celor doua polinoame
* Derivate - folosit la derivarea primului polinom
* Integrate - folosit la integrarea primului polinom
* Clear – sterge tot ce este in cele 3 zone de text

**TextField:** se poate observa că exista 3 astfel de zone, cu mentiunea faptului că cel de-al treilea text-field este setat sa nu poata fi modificat cu metoda setEditable(false). Primele doua casute sunt folosite pentru a prelua intrarea de la utilizator.

**Labels:** etichetele le-am folosit pentru a ajuta utilizatorul să știe cum să utilzeze calculatorul de polinoame.

Pentru a putea avea acces la ceea ce se scrie in textfield-urile pentru polinoame și pentru a scrie rezultatul final, am implementat cateva metode necesare:

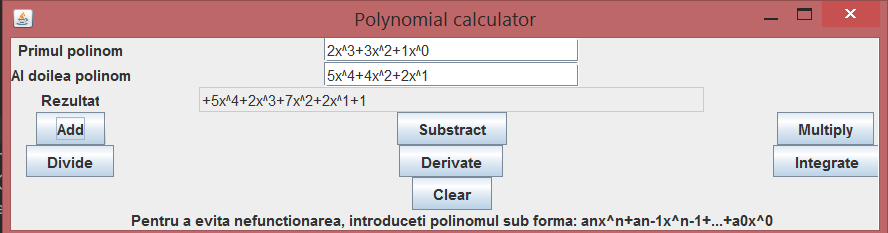
* String getUserInput1(){  
   return p1.getText();  
  }  
    
  String getUserInput2(){  
   return p2.getText();  
  }  
    
  void setResult(Polinom tot) {  
   result.setText(tot.toString());  
  }  
    
  void clear(){  
   p1.setText("");  
   p2.setText("");  
   result.setText("");  
  }

Pentru butoane, am creat metode care adauga cate un ascultator fiecarui buton care va implementa metoda actionPerformed(). Această metoda am implementat-o pentru fiecare caz in parte cu o clasa internă în clasa Controller.

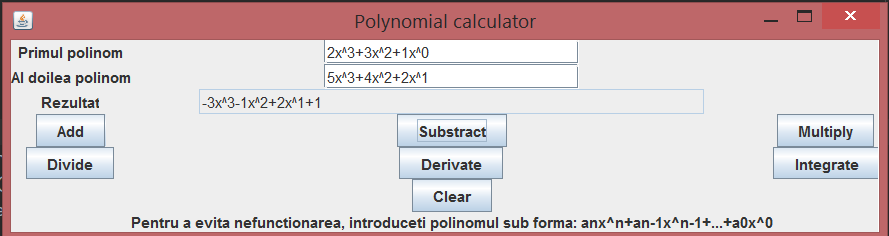
void addAddListener(ActionListener a) {  
 add.addActionListener(a);  
}

class AddListener implements ActionListener {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String in1 = view.getUserInput1();  
 String in2 = view.getUserInput2();  
 view.setResult(logic.*adunare*(convertToPolinom(in1),convertToPolinom(in2)));  
 //logic.adunare(convertToPolinom(in1),convertToPolinom(in2)).afisarePolinom();  
 }  
}

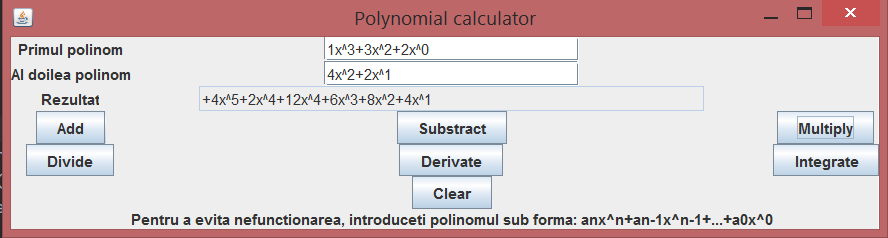
**5.Rezultate**



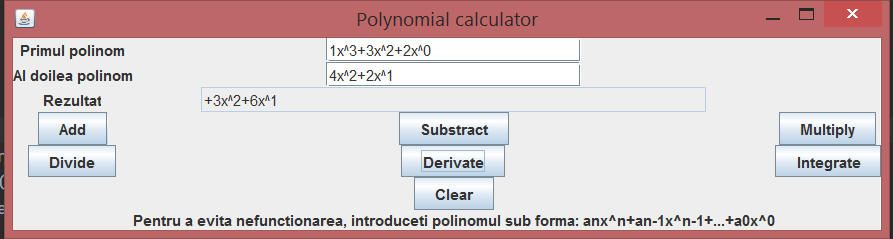
Alegerea operației de adunare:



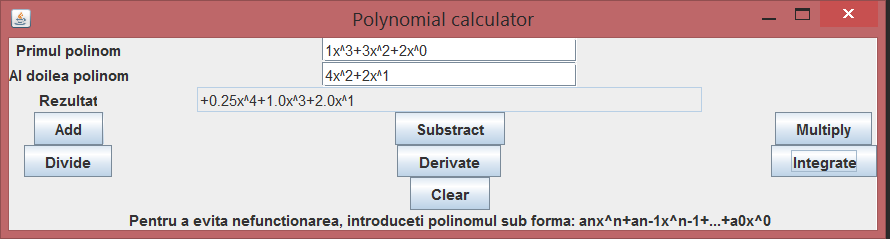
Alegerea operației de scădere:



Alegerea operației de înmulțire:



Alegerea operației de derivare (primul polinom):

Alegerea operației de integrare (primul polinom):

Interfața grafică este ”user friendly”, astfel că utilizatorul poate ușor să selecteze operația dorită și de asemenea poate vedea rezultatele dacă respectă cerințele de introducere a polinoamelor.

**6.Concluzii**

Ca o conluzie la tot proiectul, pot spune că am reușit să îmi îmbunătățesc unele cunoștințe legate de limajul de programare Java și de paradigma OOP. Consider că mai am mult de lucrat încă pe această parte, dar acest proiect a fost un început bun să fiu pusă în situația de a mă gândi la nevoile unui client.

Desigur că s-ar mai putea aduce îmbunătățiri la interfața grafică, la modificarea operației de înmulțire ca să adune și termenii care rezultă cu același grad, la finalizarea operației de împărțire și la modul în care sunt introduse polinoamele, dar acestea sunt luccruri de rezolvat în viitor.

**7. Bibliografie**

[**https://stackoverflow.com/questions/28859919/java-regex-separate-degree-coeff-of-polynomial**](https://stackoverflow.com/questions/28859919/java-regex-separate-degree-coeff-of-polynomial)

[**https://en.wikipedia.org/wiki/Polynomial\_long\_division**](https://en.wikipedia.org/wiki/Polynomial_long_division)

[**https://docs.oracle.com/javase/tutorial/essential/regex/**](https://docs.oracle.com/javase/tutorial/essential/regex/)

[**https://docs.oracle.com/javase/tutorial/uiswing/**](https://docs.oracle.com/javase/tutorial/uiswing/)

[**https://docs.oracle.com/javase/7/docs/api/**](https://docs.oracle.com/javase/7/docs/api/)