CALCULATOR DE POLINOAME

DOCUMENTAȚIE

Studentă: Aurică Alina

Materie: Tehnici de programare

Profesor îndrumător: Mitrea Dan

CUPRINS:

1. Obiectiv
2. Studiul problemei
3. Implementare

* Diagrame UML
* Clase și metode
* GUI

1. Testare
2. Concluzii
3. Bibliografie
4. OBIECTIV:

Obiectivul proiectului este de a implementa și proiecta un calculator de polinoame. Acest calculator trebuie să realizeze operațiile de adunare, scădere, înmulțire și împărțire între 2 polinoame, plus derivarea și integrarea unui polinom. Polinomul este format dintr-o singură variabila (alegem prin convenție să fie x) și să aibă coeficienți întregi.

Folosirea acestui calculator se face printr-o interfață grafică, unde utilizatorul poate să introducă manual polinoame și să selecteze operația pe care dorește să o efectueze.

1. STUDIUL PROBLEMEI:

În mod general, un polinom este format dintr-o listă de monoame. Prin definiție, un monom este un polinom singular, adică este determinat de următoarele 2 atribute: coeficient și putere. Pentru a realiza operații pe 2 polinoame, există mai multe metode de a structura codul din spate.

Am ales să lucrez pe un caz general, unde polinomul meu este de grad n, adică de forma:

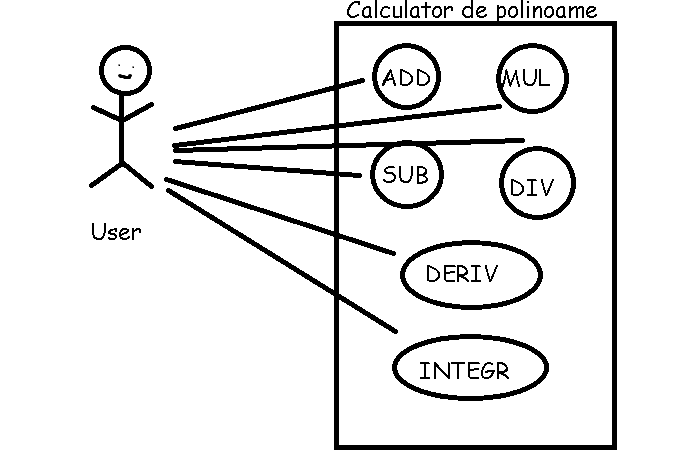
an\*xn + an-1\*xn\*1 + ... +a1\*x + a0.

Ca să extrag coeficienții și puterile (ca să pot forma un monom), am ales să folosesc un regex, prin care string-urile pe care le primesc prin intermediul TextField-urile să fie despărțite dupa +, - și ^, dar si după variabila x. Acest lucru a fost realizat într-o metodă cu același nume, în care apelez 2 alte metode statice ajutătoare:

public static Polinom regex(String exp){ //formatul prin care poate desparti regex-ul este coefX^putere  
 Polinom result = new Polinom();  
 Pattern pattern = Pattern.*compile*("([+-]?[^-+]+)");  
 Matcher matcher = pattern.matcher(exp);  
 while (matcher.find()) {  
 Monom var = new Monom(*getCoeficient*(matcher.group(1)), *getPutere*(matcher.group(1)));  
 result.addPolinom(var);  
 }  
  
 return result;  
  
}

private static int getCoeficient(String expresie){  
 String[] var = expresie.split("x");  
 int coef = Integer.*parseInt*(var[0]);  
 return coef;  
}  
  
private static int getPutere(String expresie){  
 String[] var = expresie.split("\\^");  
 int putere = Integer.*parseInt*(var[1]);  
 return putere;  
}

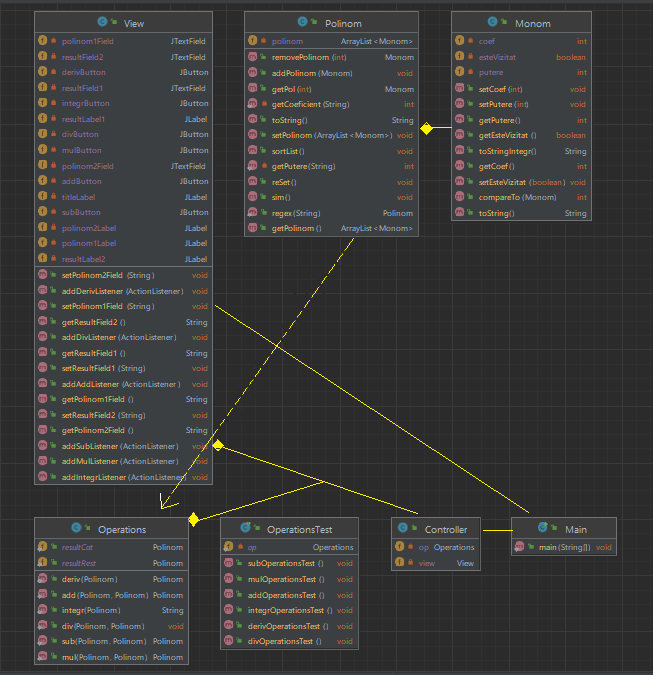
Un mod ușor de vizualizat a ceea ce este capabil să facă calculatorul este reprezentat printr-o diagramă Use Case:



unde: ADD – reprezintă operația de adunare a 2 polinoame, SUB – reprezintă operația de scădere a două polinoame, MUL - operația de înmulțire a două polinoame, DIV - operația de împărțire a două polinoame, DERIV - operația de derivare a unui polinom, iar INTEGR - operația de integrare a unui polinom.

1. IMPLEMENTARE:

* DIAGRAMA UML:



Unified Modeling Language (UML) este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost dezvoltatla început pentru reprezentarea complexității programelor care erau contruite utilizând programarea orientată pe obiect, al căror fundament este structurarea programelor pe clase și instanțele acestora (obiecte).

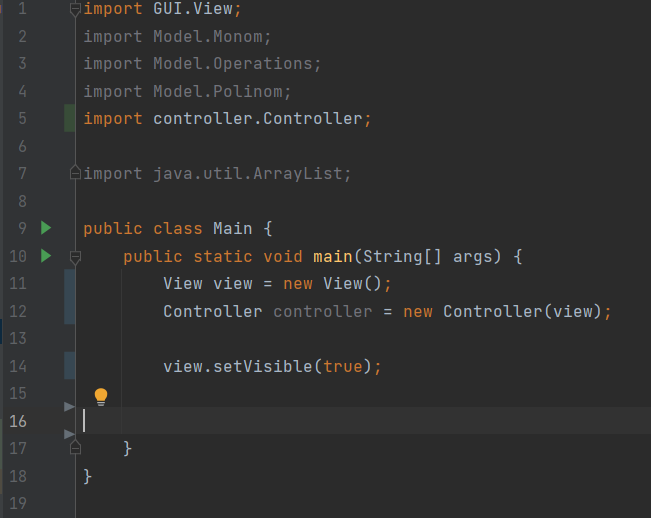
Am utilizat UML-ul pentru a vedea mult mai clar împărțirea pe clase a codului, precum și variabilele instanță și metodele pe care le-am utilizat în dezvoltarea calculatorului de polinoame.

* CLASE ȘI METODE:

Pentru structurarea codului am utilizat modelul MVC (Model, View, Controller), adică am împărțit codul în 3 pachete, la care am adăugat clasa Main.

Clasa Main controlează tot MVC-ul, având două instanțe; una a clasei View, pe care o setăm vizibilă, pentru a putea vedea interfața grafică (GUI) și cealaltă este o instanță a clasei Controller, care leagă Model-ul de View, fiind principalul punct de comandă al celor două.

Fără această clasă, care are rol unificator, utilizatorul nu s-ar putea bucura nici de interfața grafică cu utilizatorul (GUI), și nici de funcționalitățile acestui calculator de polinoame (adică nu ar putea realize operațiile de adunare, scadere, înmulțire și împărțire a două polinoame sau operațiile de derivare și integrare a unui polinom).



Model-ul conține 3 clase: clasa Monom, clasa Polinom și clasa Operations.

Clasa Monom:

Are 2 atribute de tip intreg, coef și putere, care au setată ca vizibilitate private și care sunt instanțiate în constructor:

private int coef;  
private int putere;  
private boolean esteVizitat;  
  
public Monom(int coef, int putere){  
 this.coef = coef;  
 this.putere = putere;  
}

Variabila boolean-ă esteVizitat este folosită pentru parcurgerea ArrayList-ului și stabilirea dacă acel monom a fost vizitat sau nu. Este utilizată împreuna cu metodele getEsteVizitat(); și setEsteVizitat(); în cadrul celor mai multe metode din clasa Operations.

Pentru afișarea unui obiect de tip Monom, am creat un toString() cu următorul format:

public String toString(){  
 if(this.putere > 0) {  
 if (this.coef >= 0)  
 return this.coef + "\*" + "x" + "^" + this.putere;  
 return "(" + this.coef + ")" + "\*" + "x" + "^" + this.putere;  
 }  
 else {  
 if (this.putere == 0) {  
 if (this.coef >= 0)  
 return "" + this.coef;  
 return "(" + this.coef + ")";  
 }  
 else {  
 if (this.coef >= 0)  
 return this.coef + "\*" + "x" + "^" + "(" + this.putere + ")";  
 return "(" + this.coef + ")" + "\*" + "x" + "^" + "(" + this.putere + ")";  
 }  
 }  
}

Pe lângă aceste metode, clasa Monom contine un getter și un setter pentru fiecare dintre atributele coef și putere și metoda compareTo(), ce va fi apelată în cadrul metodei sortList() din cadrul clasei Polinom:

public int compareTo(Monom m){  
 if(this.getPutere() == m.getPutere())  
 return 0;  
 if(this.getPutere() > m.getPutere())  
 return 1;  
 else return -1;  
}

Clasa Polinom:

Are un singur atribut, acesta fiind private și reprezentat de un ArrayList de Monoame. În constructor, acest ArrayList este instanțiat. De asemenea, pentru el au fost create un getter și un setter, alături de o metoda toString(), care afișează Polinomul într-un mod cât mai ușor de urmărit:

private ArrayList<Monom> polinom;  
  
public Polinom(){  
  
 polinom = new ArrayList<Monom>();  
}  
  
@Override  
public String toString(){  
 Monom e = polinom.remove(0);  
 String result = " ";  
 result = result + e.toString();  
  
 for(Monom e1:polinom){  
 result = result + " + " + e1.toString();  
 }  
  
 polinom.add(e);  
 return result;  
}

Clasa Polinom utilizează metodele add() și remove() ale ArrayList-ului, în două metode numite addPolinom() și removePolinom() care au rolul de a introduce și șterge monoame din polinoame.

public void addPolinom(Monom monom){  
  
 this.polinom.add(monom);  
}  
  
public Monom removePolinom(int i){  
  
 return this.polinom.remove(i);  
}

Tot în clasa Polinom am o metodă ce îmi sortează lista de monoame și anume sortList(). Acestă metodă este folosită cu precădere în metoda deriv() a clasei Operations.

public void sortList(){ //nu merge sortarea cum trebuie  
 for(Monom p1:this.polinom){  
 for(Monom p2:this.polinom) {  
 if (p1.compareTo(p2) == -1 && p2.getEsteVizitat() == false) {  
 Monom var = new Monom(p1.getCoef(), p1.getPutere());  
 p1.setCoef(p2.getCoef());  
 p1.setPutere(p2.getPutere());  
 p2.setCoef(var.getCoef());  
 p2.setPutere(var.getPutere());  
 }  
 }  
 p1.setEsteVizitat(true);  
 }  
}

O altă metodă utilizată în cadrul clasei Operations, cu precădere în metoda mul() este sim() care caută să restrângă un rezultat după puteri. Spre exemplu, dacă avem polinomul 3x2+5x+7+4x, noi îl dorim de forma 3x2+9x+7.

public void sim(){  
 Polinom result = new Polinom();  
  
 for(Monom p1 : this.getPolinom()){  
 for(Monom p2 : this.getPolinom()){  
 int i = this.getPolinom().indexOf(p1);  
 int j = this.getPolinom().indexOf(p2);  
  
 if(p1.getPutere() == p2.getPutere() && i != j && p1.getEsteVizitat() == false){  
 p1.setEsteVizitat( true );  
 p2.setEsteVizitat( true );  
 Monom var = new Monom(p1.getCoef() + p2.getCoef(), p1.getPutere());  
 result.addPolinom(var);  
 }  
 }  
 }  
  
 for(Monom p1 : this.getPolinom()){  
 if(p1.getEsteVizitat() == false){  
 result.addPolinom(p1);  
 }  
 }  
  
 this.setPolinom(result.getPolinom());  
}

Clasa Operations:

Toate metodele acestei clase sunt metode statice, deoarece ne dorim ca acestea să fie apelate prin intermediul clasei.

ADUNAREA:



Adunarea funcționează după următorul principiu:

Parcurgem cu ajutorul unor for-each-uri polinom1 și polinom2. Atunci când p1 și p2 (două monoame corespunzătoare polinoamenlor polinom1 și polinom2) au puteri egale, acestea sunt setare ca vizitate. Se creează o variabilă ajutătoare var căreia îi setam coef = p1.getCoef() + p2.getCoef(), iar puterea va fi puterea monomului p1. Această variabilă var este adăugată în polinomul result. Ultimele 2 for-uri sunt folosite pentru a adăuga în rezultat monoamele rămase nevizitate în cele 2 polinoame.

Ex: polinom1 = 5x5+6x2+1, polinom2 = 6x2+x

Result = 5x5+12x2+x+1

SCĂDEREA:

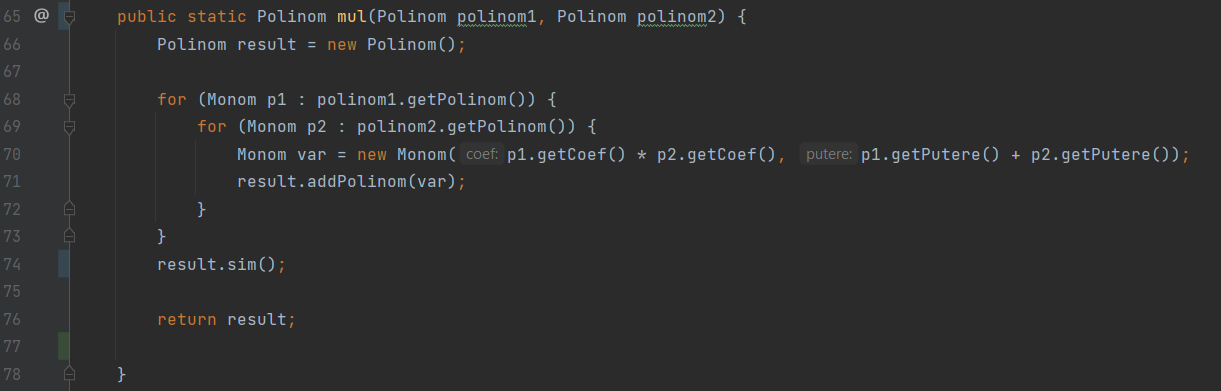


Funcționează pe același principiu ca și adunarea, singura diferentă este ca acum coeficienții se scad dacă puterile celor două monoame, p1 și p2, sunt egale.

Ex: polinom1 = 5x5+6x2+1, polinom2 = 6x2+x

Result = 5x5-x+1

ÎNMULȚIREA:



Principiul după care funcționează înmulțirea este următorul:

Cele două polinoame sunt parcurse prin for-each-uri, înmulțindu-se fiecare p1 din polinom1, cu fiecare p2 din polinom2, adică într-o variabilă var se înmulțesc coeficienții și se adună puterile. Metoda sim() restrânge rezultatul.

ÎMPĂRȚIREA:

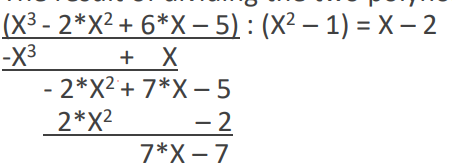
public static Polinom *resultCat* = new Polinom();  
public static Polinom *resultRest* = new Polinom();  
  
public static void div(Polinom polinom1, Polinom polinom2) {  
 polinom1.reSet();

polinom2.reSet();  
 polinom1.sortList();

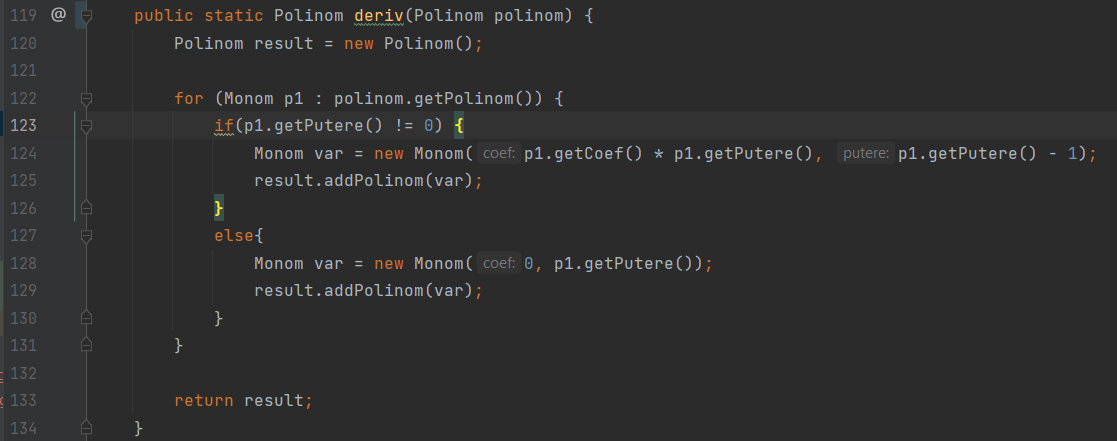
polinom2.sortList();  
   
 int i = polinom1.getPolinom().size()-1;  
 int j = polinom2.getPolinom().size()-1;  
 Monom monom1 = polinom1.getPol(i); Monom monom2 = polinom2.getPol(j);  
 Monom var = new Monom(monom1.getCoef(), monom1.getPutere() - monom2.getPutere());  
 Polinom intermedVar = new Polinom();  
 intermedVar.addPolinom(var);   
 *resultCat*.addPolinom(var);  
 System.*out*.println("cat: " + *resultCat*.toString());  
  
 Polinom intermedMul = Operations.*mul*(polinom2, intermedVar); //  
 polinom1.reSet(); intermedMul.reSet();  
 Polinom intermedSub = Operations.*sub*(polinom1, intermedMul);  
  
 polinom1.setPolinom(intermedSub.getPolinom());  
 polinom1.sortList();  
 polinom1.removePolinom(0);  
  
 polinom2.reSet();  
 polinom2.sortList();  
 if(polinom1.getPol(0).getPutere() >= polinom2.getPol(0).getPutere()) {  
 Operations.*div*(polinom1, polinom2);  
 }  
  
 *resultRest*.setPolinom(polinom1.getPolinom());  
}

Principiul după care funcționează împărțirea este următorul:

Se sortează mai întâi cele 2 polinoame în ordine descrescătoare. Se iau din fiecare monoamele cu puterea cea mai mare. Într-o variabilă ajutătoare var, se setează coef ca fiind coeficientul monomului din primul polinom, iar puterea ca diferenta dintre puterea monomului din primul polinom și puterea monomului celui de-al doilea polinom. Se adaugă această varianbilă în resultCat (variabilă statică, declarată înafara metodei). Se formează noul polinom1 ca difernta dintre cel vechi și produsul dintre polinom2 și resultCat. În if se verifică dacă puterea maximă a lui polinom1 e mai mare decât puterea maximă a lui polinom2. Dacă da, se apelează recursiv metoda div(), în caz contrar, se trece direct la formarea polinomului resultRest, care reprezinta restul împărțirii.

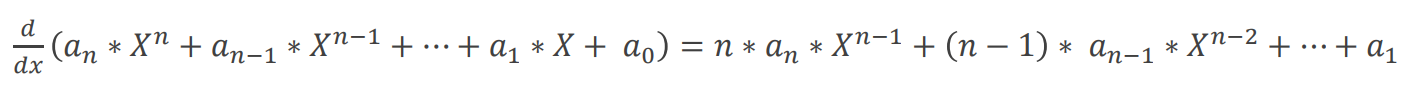


DERIVAREA:

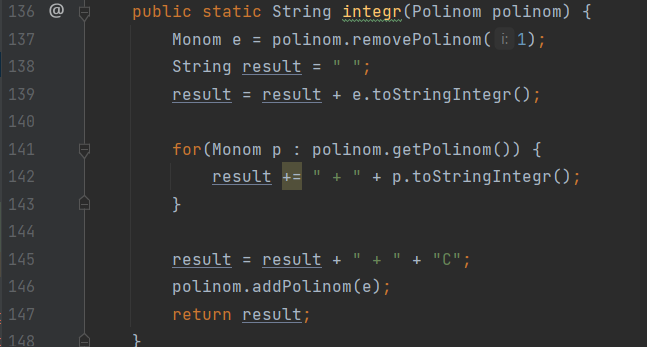


Derivarea are următorul principiu de funcționare:

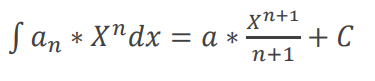
Polinomul este parcurs printr-un for-each. Pentru fiecare monom al acestuia, avem o variabilă ajutătoare var, care are ca și coeficient, produsul dintre coeficientul și puterea monomului p1, iar ca putere, puterea monomului p1 – 1.



INTEGRAREA:



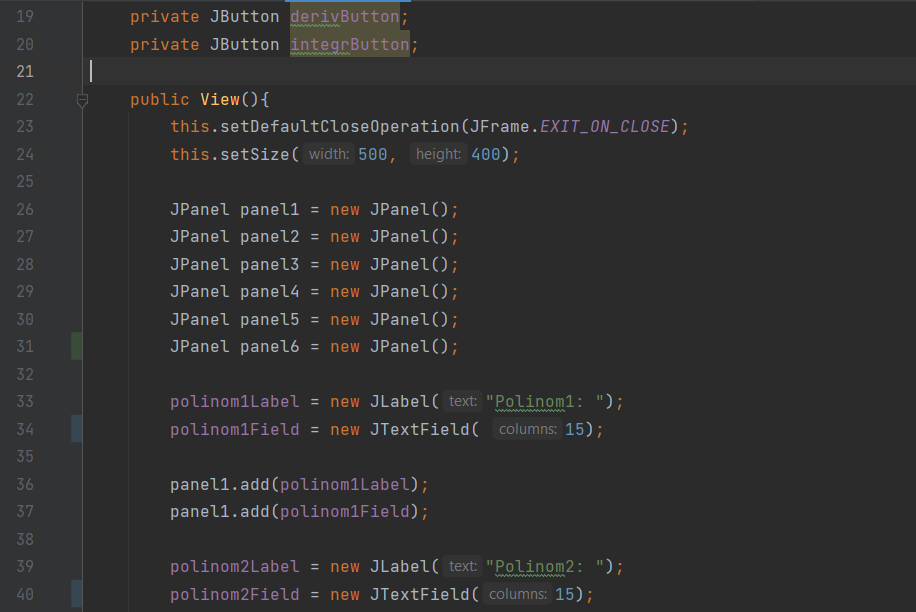
Integrarea este realizată ca un toString, deoarece am vrut să respect următorul format:



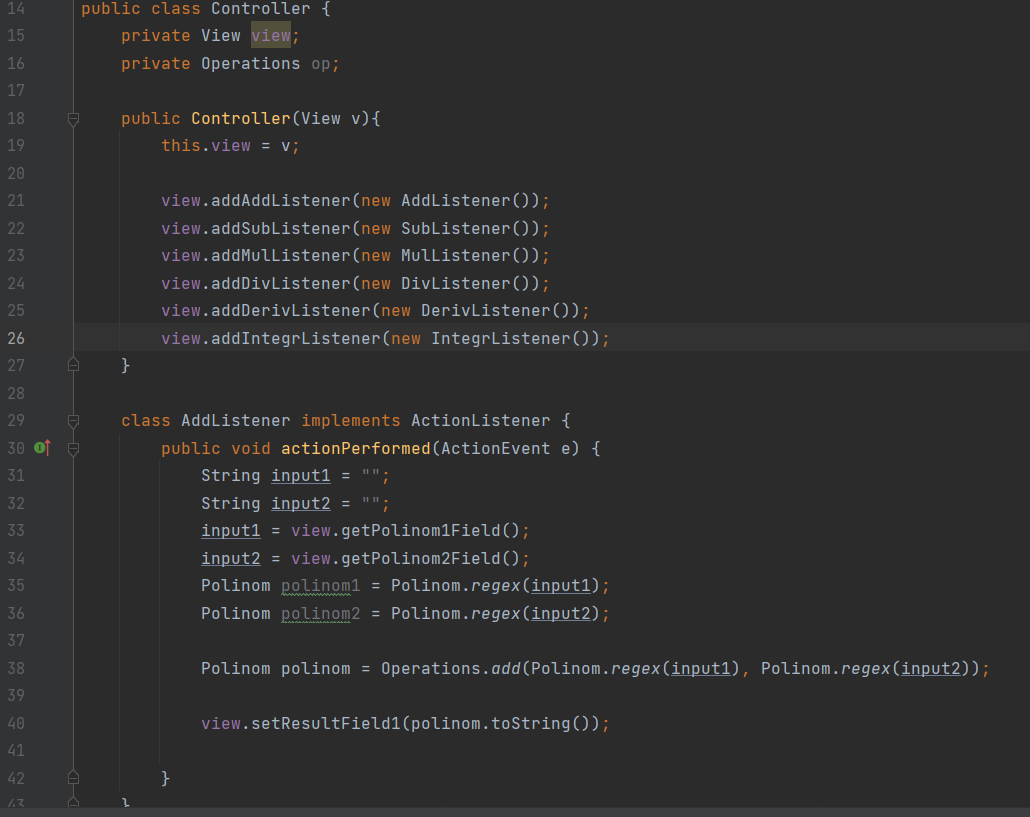
* GUI

Pentru crearea GUI-ului (Interfeței grafice cu utilizatorul), am împărțit implementarea în Controller și View. Clasa Controller conține toate clasele necesare funcționării butoanelor, iar clasa View conține codul din spatele proiecției fiecărui element în Jframe.

Aceasta este o porțiune din clasa View, care conține getter-ere și setter-ere pentru fiecare JtextField și addListener() pentru fiecare buton:



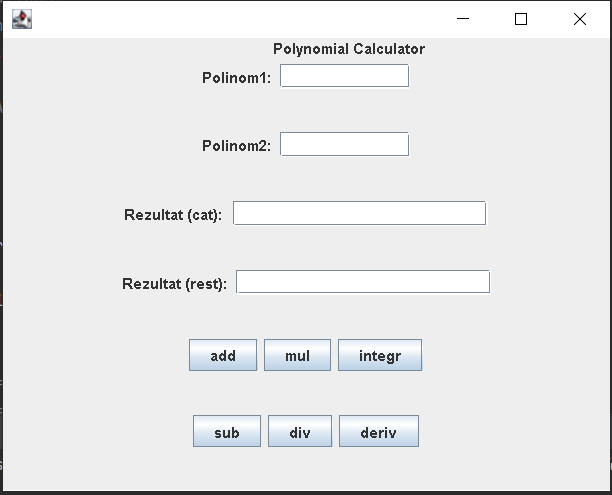
Aceasta este o porțiune din clasa Controller, care conține o clasă internă pentru fiecare buton:



Constructorul clasei Controller este special, deoarece primește ca parametru un obiect al clasei View și instanțiaza toate metodele obiectului view care fac referire la butoane.

Pentru a realiza funcționalitatea butoanelor, trebuie să implementăm interfața ActionListener, care are o metodă, actionPerformed(), ce trebuie implementată (practic, suprascrisă) în fiecare clasă ce folosește acestă interfață.

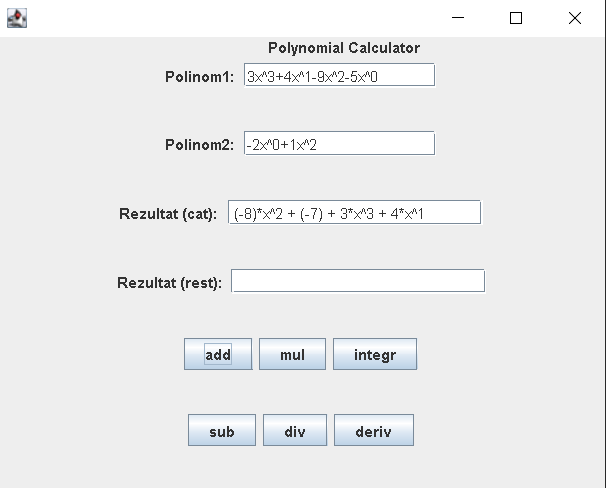
Aceasta este interfața grafică pe care o vizualizează utilizatorul:



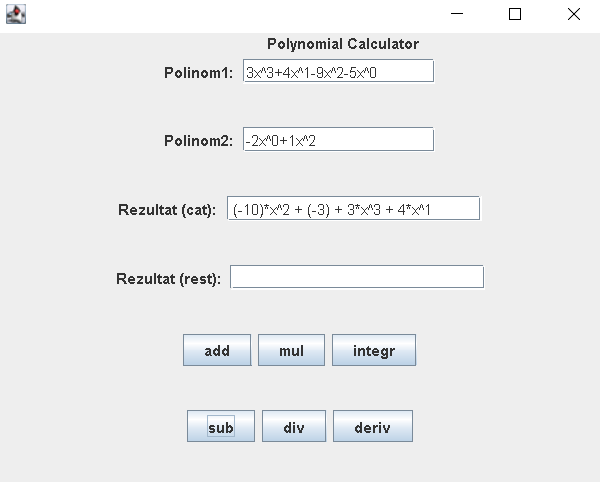
Ea conține 6 butoane, fiecare dintre ele apelând una din operațiile implementate în clasa Operations. De asemenea, avem 4 TextField-uri, primele 2 sunt pentru a introduce polinoamele cu care lucrăm, următoarele 2 sunt pentru a afișa rezultatul obținut în urma operațiilor (amândouă se folosesc doar pentru operația de împărțire). În fața fiecărul TextField, pentru o vizualizare mult mai bună, sunt adăugate Label-uri. Acestea funcționează ca niște etichete. Tot printr-un astfel de Label am adăugat și un titlu aplicației.

Întreaga interfață este struncturată în 6 JPanel-uri, pentru a obține acest aspect (pentru ca fiecare TextFiel și buton să fie unul sub altul). Toate aceste panel-uri au fost adăugate într-un al 7-lea JPanel, care este inclus la rândul său într-un JFrame.

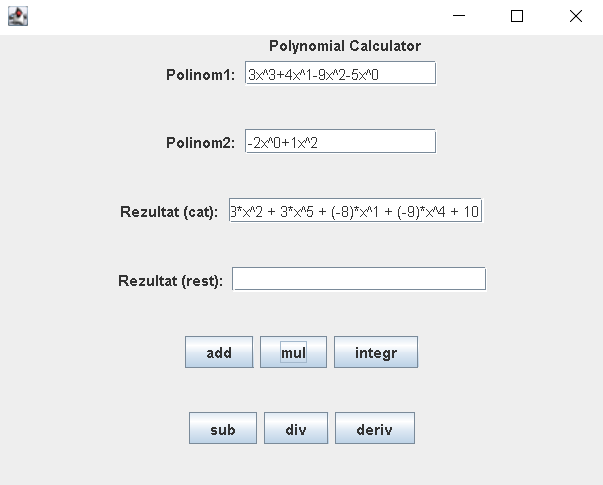
Prin apăsarea butonului add:



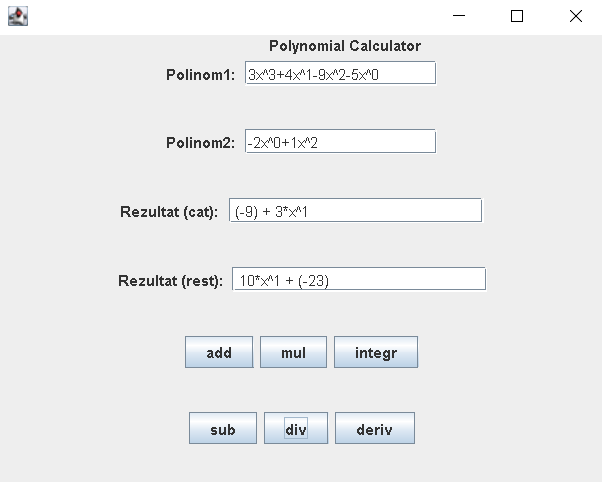
Prin apăsarea butonului sub:



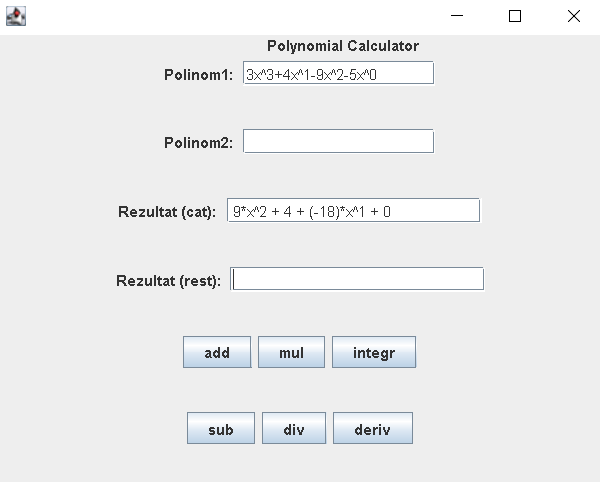
Prin apăsarea butonului mul:



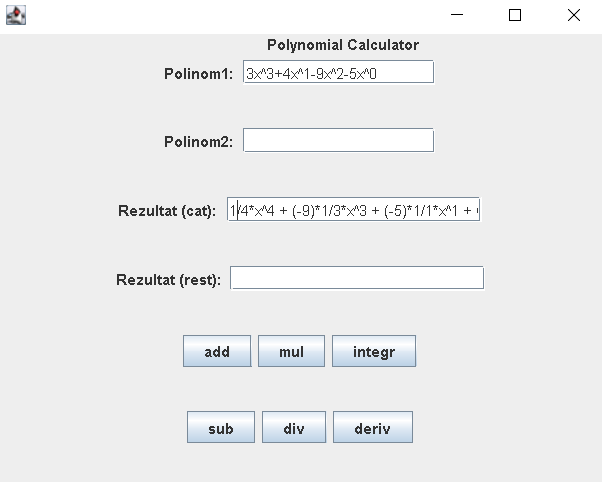
Prin apăsarea butonului div:



Prin apăsarea butonului deriv:



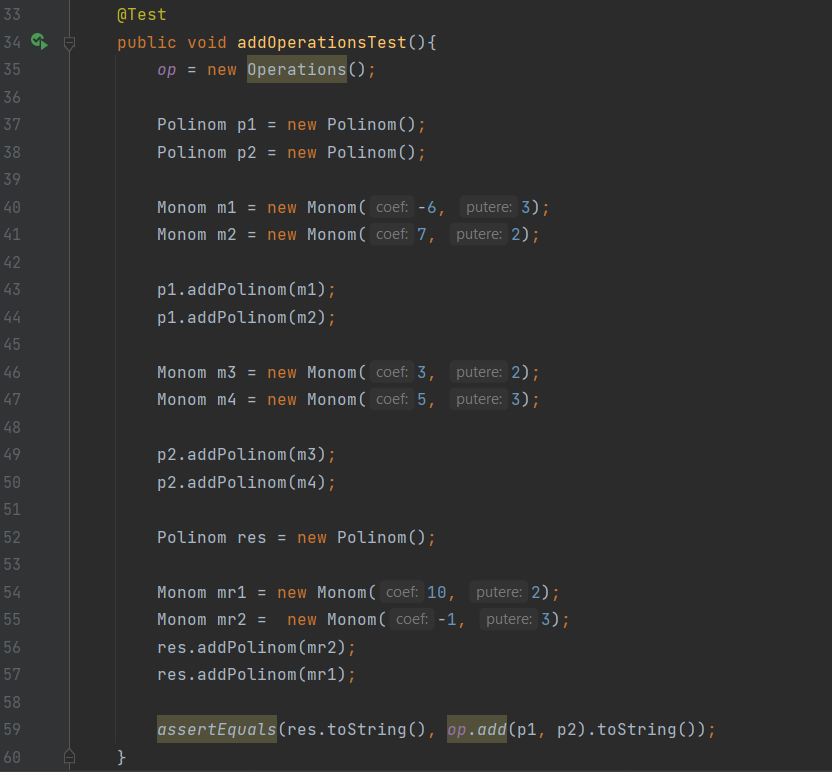
Prin apăsarea butonului integr:



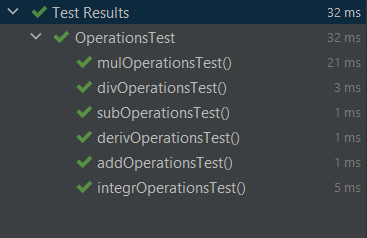
1. TESTARE:

Partea de testare a proiectului a fost realizata cu ajutorul bibliotecii Junit. Practic, am creat o clasă unde cu ajutorul metodei assertEquals, am verificat daca adunarea, scaderea, înmulțirea, împărțirea, derivarea și integrarea unor polinoame, corespunde cu un polinom result, pe care îl setăm noi cu rezultatul pe care îl știm ca fiind adevărat.

Acesta este una dintre metodele clasei OperationsTest:



Pe același principiu se bazează și celelalte metode de test.



După cum se poate observa, toate cele 6 teste sunt executate cu succes.

1. CONCLUZII:

Concluzia acestui proiect pe care tocmai l-am realizat este că un astfel de dispozitiv ușurează foarte mult viața oamenilor care lucrează în domenii științifice, precum matematica, fizica și chiar ingineria.

În ceea ce mă privește, proiectul mi-a oferit oportunitatea de a-mi dezvolta lucrul cu limbajul Java, cu paradigma programării orientate pe obiect. Mi-a oferit oportunitatea de a crea un proiect complex, ce utilizează numeroase librării, precum: JUnit, javax.swing și java.awt. De asemenea, m-a ajutat să-mi reamintesc tehnicile de programare învățate semestrul trecut.

1. BIBLIOGRAFIE:

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

<https://stackoverflow.com/>

<https://www.geeksforgeeks.org/>