**DOCUMENTAȚIE TEMA 1**

**CALCULATOR POLINOAME**

**Nume prenume: Copoț Raluca**

**Grupa: 30227**

**Profesor laborator: Mitrea Dan**

Cuprins

1. **Obiectivul temei3**
   1. Obiectivul principal3
   2. Obiective secundare3
2. **Analiza problemei3**

2.1 Analiza principală 4

2.2 Cerințe funcționale 4

2.3 Cerințe non-funcționale 4

1. **Proiectare5**

3.1 Determinarea structurilor de date5

3.2 Împărțirea în clase și pachete5

3.3 Algoritmi6

1. **Implementare7**

4.1 Clase și metode7

4.2 GUI9

1. **Rezultate10**

5.1 Funcționare10

5.2 Testare cu JUnit11

1. **Concluzii11**
2. **Bibliografie12**
3. **Obiectivul temei**
   1. **Obiectivul principal**

Proiectarea și implementare unui calculator de polinoame de o singura variabila, cu coeficienți întregi.

* 1. **Obiective secundare**

|  |  |  |
| --- | --- | --- |
| **Obiectiv secundar** | **Descriere** | **Capitol** |
| Analizarea problemei și identificarea cerințelor | Realizarea diagramei use-case și determinarea scenariilor posibile | 2 |
| Determinarea structurilor de date | Deoarece nu există un tip predefinit de data de tipul polinom, acesta trebuie creat și implementat | 3 |
| Împărțirea în clase si pachete | Determinarea unui scop bine definit pentru fiecare clasă și distribuirea acestora în diferite pachete, în funcție de rol | 3 |
| Determinarea și dezvoltarea algoritmilor | Un algoritm trebuie dezvoltat și implementat pentru fiecare operație matematică dintre polinoame | 3 |
| Implementarea calculatorului de polinoame | Scrierea propriu-zisă a codului, implementarea efectivă a algoritmilor determinați anterior | 4 |
| Testarea | Verificarea corectitudinii calculatorului de polinoame pentru a vedea daca exista erori sau cazuri netratate | 5 |

1. **Analiza problemei**

**2.1. Analiza principală**

Exemplu de use-case pentru operația de adunare:

**Use case**: adunarea polinoamelor

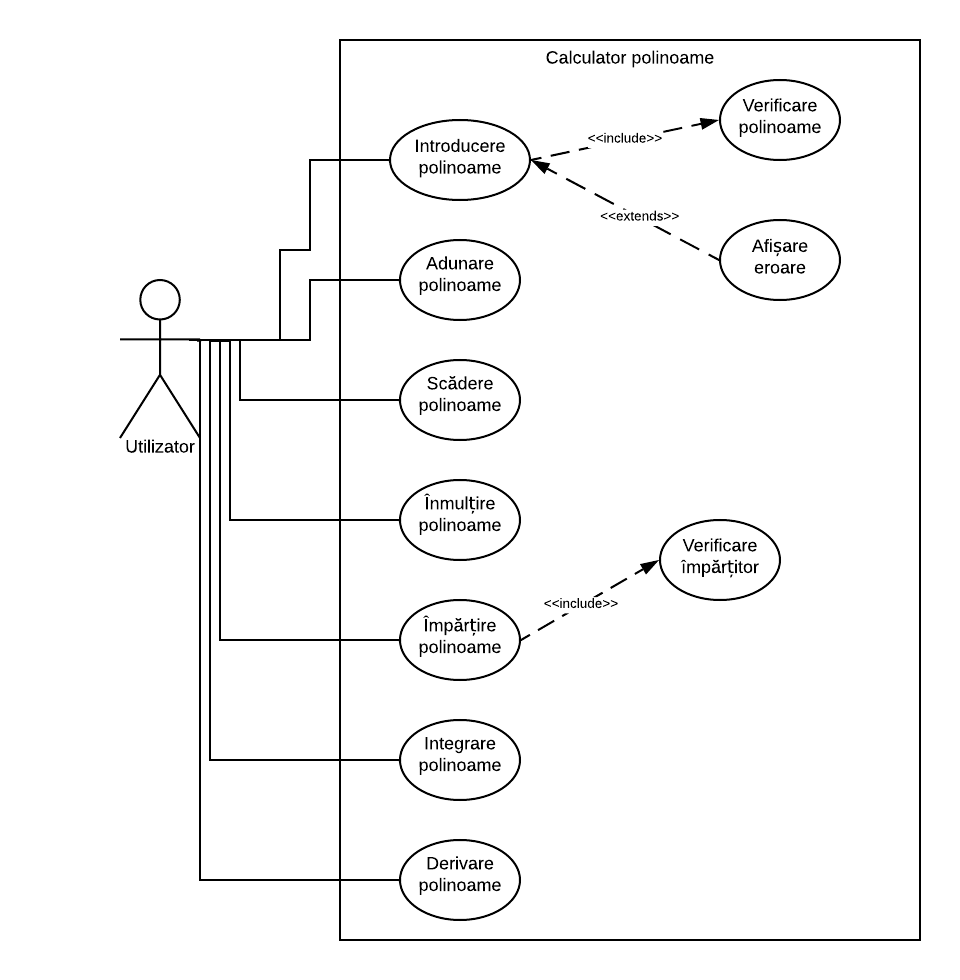
**Actorul primar**: utilizatorul

**Scenariul principal în caz de succes**:

1. Utilizatorul introduce cele doua polinoame în text-field-urile din interfața grafică.
2. Utilizatorul apasă pe butonul corespunzător operației pentru adunare.
3. Calculatorul efectuează operația de adunare a celor două polinoame și afișează rezultatul text-field-ul corespunzător.

**Scenariul alternativ**: Introducerea unui polinom invalid

* Utilizatorul introduce un polinom care nu are un format valid sau un cuvânt la întâmplare
* Aplicația afișează un mesaj de eroare
* Scenariul se întoarce la pasul 1

**Diagrama use case:**

**2.2. Cerințe funcționale**

Proiectați și implementați un calculator de polinoame cu o interfață grafică, prin care utilizatorul poate efectua operațiile:

* Citirea unui polinom de la tastatură;
* Selectarea și efectuarea operaților de : adunare, scădere, înmulțire, împărțire, derivare, integrare;
* Revenirea la starea inițială a calculatorului de polinoame;

**2.3. Cerințe non-funcționale**

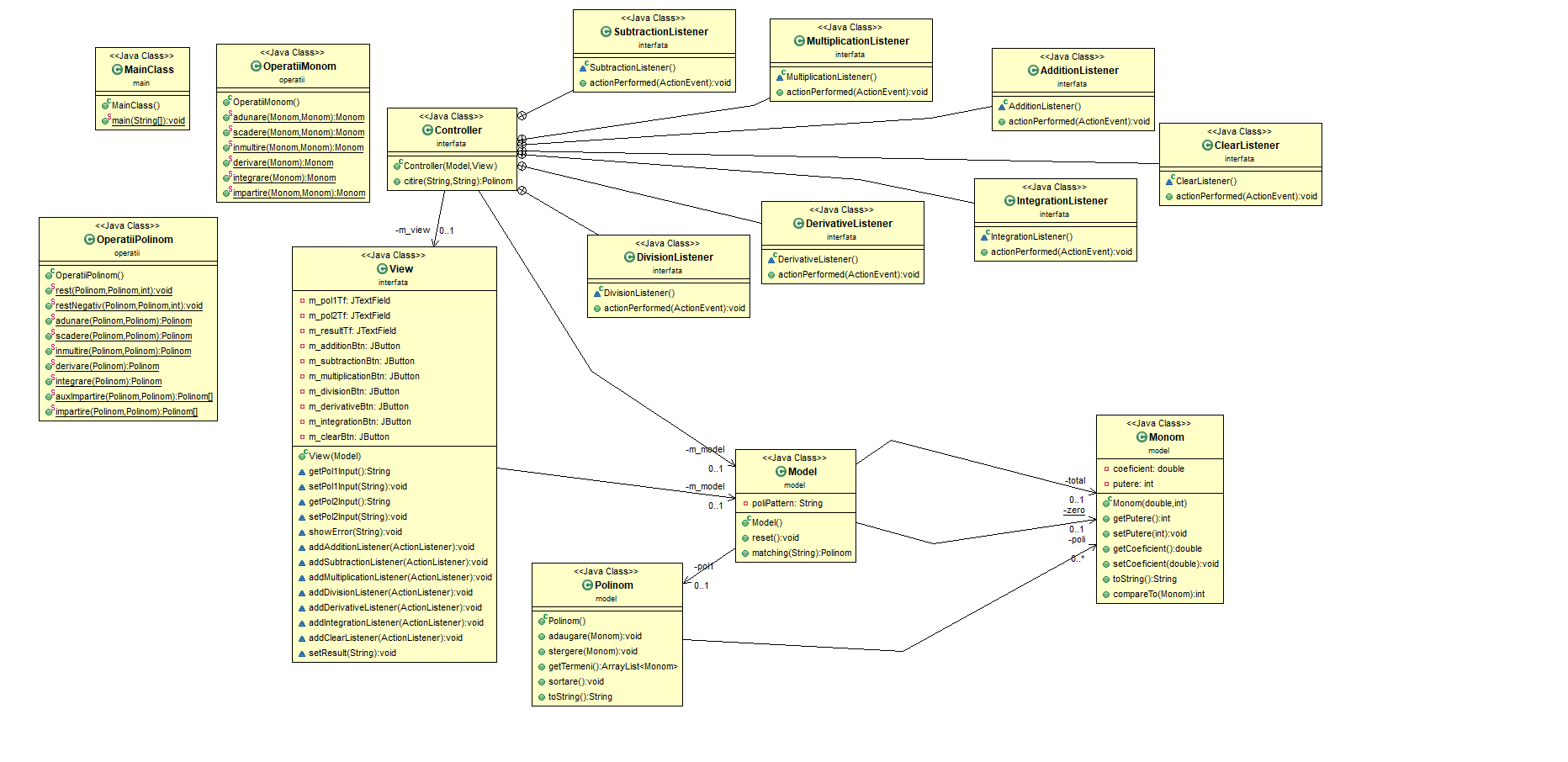
* Calculatorul de polinoame ar trebui să fie intuitiv și ușor de folosit de utilizator;
* Designul calculatorului ar trebui să fie unul compact, dar care conțină toate elementele dorite;
* Calculatorul ar trebui sa atenționeze utilizatorul atunci când introduce un polinom invalid și să nu efectueze calculele decât după ce primește inputul corespunzător;
* Rezultatul afișat de calculator ar trebui sa fie lizibil, ușor de înțeles și interpretat ;

1. **Proiectare**
   1. **Determinarea structurilor de date**

Deoarece tipul de date sau obiectul de tip „polinom” nu este unul predefinit în Java, acesta trebuie implementat. Am ales a doua variantă, cea de obiect, însă pentru a face acest lucru am analizat structura unui polinom, acesta fiind format din mai multe monoame. La rândul lor, monoamele conțin un coeficient, care ar trebui să fie real pentru ușurință în cazul împărțirii, respectiv al integrării, necunoscuta, „x”, care nu este necesară în implementare, deoarece în cazul acesta polinoamele au o singură variabilă, și exponentul necunoscutei, care este un număr întreg.

Deși este format din mai multe monoame, polinomul nu poate fi o sumă a acestora, cum este cazul din punct de vedere matematic. În plus, nu poate fi nici un vector, array de monoame, fiind dificil și ineficient de implementat. Așadar, am considerat polinomul ca fiind un arrayList de monoame, parcurgerea, adăugarea și ștergerea devenind totodată mai ușor de realizat.

* 1. **Împărțirea în clase și pachete**

**Diagrama UML:**

Â

După ce am determinat tipul de date folosit, urmează împărțirea în clase, respectiv în pachete. Două dintre acestea reies după analiza de la pasul precedent, acestea fiind clasele Monom, respectiv Polinom. De asemenea, folosind pattern-ul arhitectural MVC, este nevoie și de clasele Model, View și Controller, care își au rolul în implementarea interfeței grafice. Pentru efectuarea operațiilor am ales să creez două clase OperatiiMonom și OperatiiPolinom, având astfel abilitatea de a realiza atât operații între monoame, cât și între polinoame. În ultimul rând, avem nevoie de MainClass.

Pentru împărțirea în pachete, am folosit ca și criteriu funcționalitatea fiecărei clase. Așadar, am ajuns la pachetele:

* interfata, în care se află clasele care au ca rol implementarea acesteia, View și Controllerș
* model, care conține clasele care conțin logica de bază din spatele aplicației, Model, Monom și Polinom
* operatii, după cum sugerează numele, aici se afla OperatiiPolinom și OperatiiMonom
* main, care conține clasa MainClass
  1. **Algoritmi**

**Adunarea:** Începem parcurgerea celor două polinoame pornind de la termenii cu cea mai mare putere. În cazul favorabil, în care aceștia au aceeași putere atunci adunăm monoamele, lucru realizat prin însumarea coeficienților.Dacă, în schimb, avem un monom cu o putere mai mare, îl adăugam în polinomul rezultat si trecem la următorul termen**.** Această parcurgere este repetată până când parcurgem toți termenii din ambele polinoame.

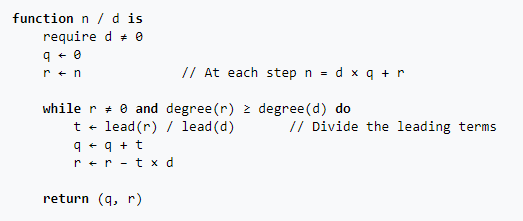
**Scăderea:** Asemenea adunării, diferența fiind în faptul ca scădem coeficienții. De asemenea, dacă unul dintre monoamele celui de-al doilea polinom, al scăzătorului, are gradul dominant, atunci îl vom înmulți cu minus unu înainte de a îl adăuga în polinomul rezultat.

**Înmulțirea:** Pentru elementele de tipul monom, înmulțirea constă în adunarea exponenților și înmulțirea coeficienților. Pentru polinoame, fiecare element din primul operand va fi înmulțit cu fiecare element din al doilea operand, rezultatul produsului fiind stocat într-un monom auxiliar. Acesta se va aduna cu polinomul rezultat, inițial lipsit de elemente, evitând, astfel existența mai multor termeni cu același exponent.

**Integrarea:** Folosindu-ne de formula matematică, în cazul obiectelor de tipul monom, integrarea constă în incrementarea exponentului și în împărțirea coeficientului, astfel: coeficient = coeficient/(1+exponent). Pentru polinom, aplicăm operația de integrarea a monoamelor tuturor elementelor sale.

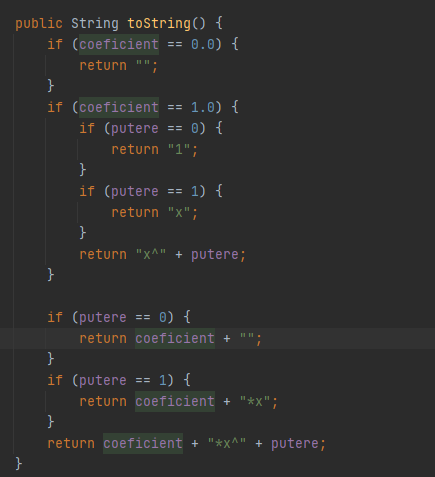
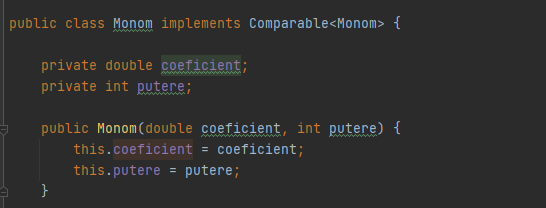
**Derivarea:** Precum în cazul integrării, ne folosim de formulele din analiza matematica pentru a determina un algoritm pentru elementele de tip monom. În acest caz, exponentul este decrementat, iar coeficientul devine rezultatul produsului dintre vechiul coeficient si puterea înainte de decrementare. Din nou, aplicăm această operațiile tuturor elementelor din polinom pentru a obține rezultatul final.

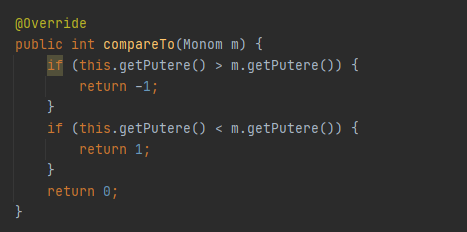
**Împărțirea:** În primul rând, determinăm care este polinomul cu un grad mai mare, acesta fiind deîmpărțitul. Verificăm apoi ca împărțitorul să nu fie nul. Dacă aceste condiții sunt îndeplinite, cât timp deîmpărțitul este diferit de zero și gradul său este mai mare decât al împărțitorului, stocăm într-un monom auxiliar rezultatul împărțirii monoamelor de grad maxim ale celor doua polinoame. Îl adunăm apoi polinomului rezultat și scădem din deîmpărțit valoarea produsului dintre polinomul auxiliar și împărțitor. Ceea ce va rămâne la sfârșitul parcurgerii din primul polinom reprezintă restul împărțirii. Pentru monoame, împărțirea constă în scăderea exponenților și împărțirea coeficienților. Pseudocodul care stă la baza algoritmului este următorul:



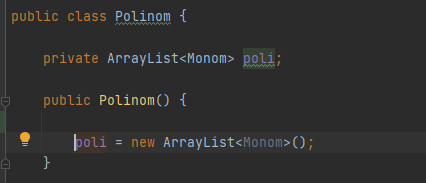
1. **Implementare**
   1. **Clase și metode**

**Monom:** conține structura unui monom. Din analizele realizate anterior, am decis ca acesta să aibă un coeficient de tip double, pentru a nu fi necesară reprezentarea rezultatului împărțirii și a integrării sub formă de fracții, și o putere, care este un număr întreg. Această clasă implementează interfața Comparable<T>, pentru a putea suprascrie metoda compareTo, sortând monoamele descrescător, în funcție de exponent. În clasă este suprascrisă și metoda toString, pentru a putea afișa rezultatul într-un format care poate fi înțeles de utilizator. De asemenea, am implementat și metode de get, set și bineînțeles, un constructor.

****

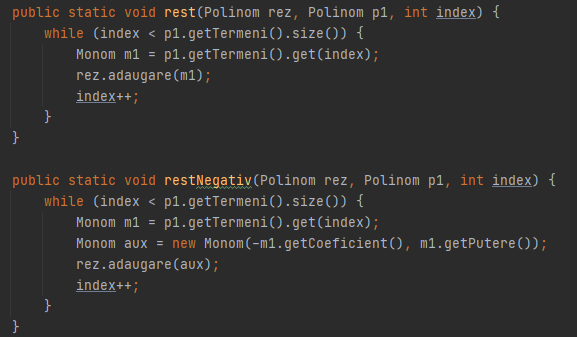
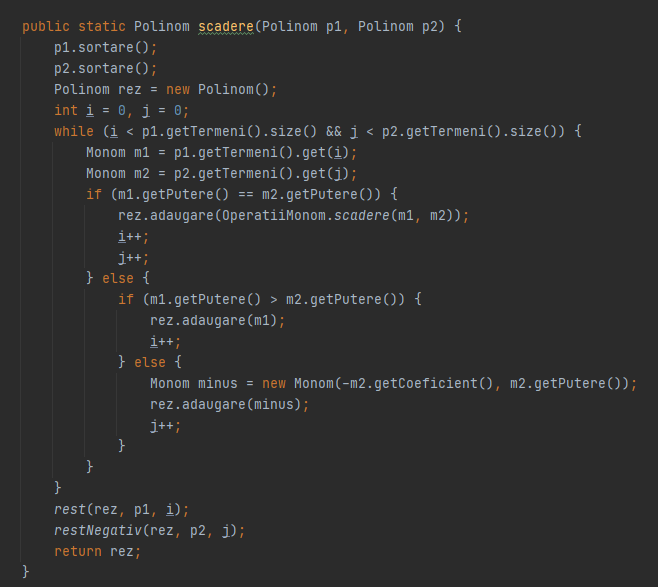
****

**Polinom:** conține structura unui polinom, pe care am ales să în implementez precum un ArrayList de monoame. În afară de constructor, în această clasă am implementat metode de adăugare, respectiv ștergere a unui element din listă, o metodă care returnează toate elementele, având nevoie de aceasta pentru parcurgerile efectuate în celelalte clase, dar și o metodă care va sorta elementele, apelând compareTo din clasa Monom. În plus, am suprascris și în această clasă toString-ul. Aceasta îmi va adăuga monoamele într-un string rezultat și semnul plus dacă avem un coeficient pozitiv, respectiv doar un spațiu daca e unul negativ. String-ul rezultat va fi ulterior afișat.

****

**Model:** în această clasă am implementat regex-ul. Unul dintre cele mai importante atribute ale aceste clase este pattern-ul, care ma ajută să determin dacă șirul de caractere introdus de utilizator este un polinom. Pattern-ul este, așadar de forma: iar metoda matching este cea în care se afla regex-ul propriu-zis. Cât timp există o potrivire, luăm fiecare caracter din grupul care este de forma șablonului poliPattern. Dacă este o cifră sau un semn, atunci în adăugam într-un String auxiliar, numit curent. În cazul în care este necunoscuta, care poate fi doar litera „x”, aplicam parseInt, rezultatul fiind coeficientul. Am tratat și cazul în care ceoficientul este implicit 1, caz în care adăugăm aceasta valoare șirului curent înainte de a face parse. Dacă nu întâlnim caracterul „x” și ajungem la finalul grupului, exponentul va lua valoarea din String, după un parseInt. Iar dacă la finalul parcurgerii coeficientul are valoarea zero, acest lucru semnifică faptul că avem un termen liber. Fiecare dintre aceste grupuri reprezintă un monom, care va fi adăugat polinomului rezultat.

**OperatiiMonom:** existența acestei clase este motivată de nevoia de a nu avea metode cu multe linii de cod și de cea de a putea testa cu ușurință în timpul implementării, în „main”, funcționarea corectă. Metodele din această clase sunt statice, ele fiind apelate în interiorul metodelor din clasa „OperatiiPolinom”. Acestea sunt „adunare”, „scadere”, în interiorul cărora se vor aduna, respectiv scădea coeficienții doar în cazul în care cele doua polinoame au același grad. Va fi returnat un monom cu acest nou coeficient si puterea polinoamelor. În caz contrar se va returna null. Clasa conține și metodele „integrare”, „derivare” și „inmultire”, care implementează logica menționată anterior, în secțiunea în care sunt explicați în cuvinte algoritmii acestor operații. Pentru metoda „impartire”, în primul rând în cazul în care coeficientul împărțitorului este zero, metoda va returna null. Altfel, se va efectua împărțirea după regula precizată anterior. Toate aceste metode returnează, în caz de succes, un monom, cu exponentul și coeficientul calculate în interiorul metodei respective.

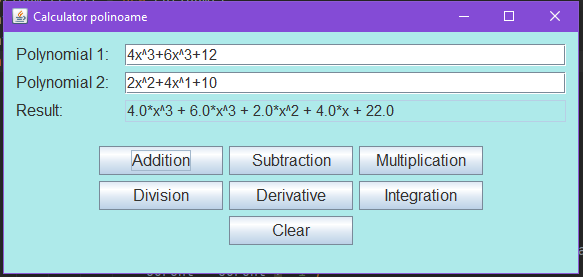
******OperatiiPolinom:** și în această clasă metodele sunt statice, ele implementând, după cum sugerează numele, operațiile dintre polinoame. Metoda adunare sortează, pentru început cele două polinoame, urmând ca ele să fie parcurse. Deoarece am nevoie să pot trece la termenul următor fie pentru ambele polinoame, fie doar pentru unul, nu pot folosi foreach, motiv pentru care am ales bucla while. Pentru monoamele de același grad apelez metoda statică de adunare din clasa Monom, iar daca gradele nu coincid, cel de grad maxim este adăugat în polinomul rezultat. Iar, pentru că există cazuri în care nu sunt parcurși toți termenii in aceasta parcurgere inițială, apelez metoda rest, care adaugă eventualii termeni care au fost omiși, în polinomul rezultat. Metoda scădere este asemanătoare, diferența majoră fiind în faptul ca, în acest caz, înmulțesc cu minus unu termenii din al doilea polinom înainte de a îi adăuga. Metodele „inmultire”, „derivare” și „integrare” parcurg toți termenii și apelează metoda corespunzătoare din clasa Monom, după cum am precizat anterior. Pentru a generaliza împărțirea am implementat metoda „auxImpartire”, care determină polinomul de grad maxim si apelează metoda „impartire” avându-l pe acesta ca prim termen. Cele două metode folosite pentru împărțire returnează un vector de doua polinoame, primul fiind câtul, iar al doilea restul. Celelalte metode returnează un singur polinom.

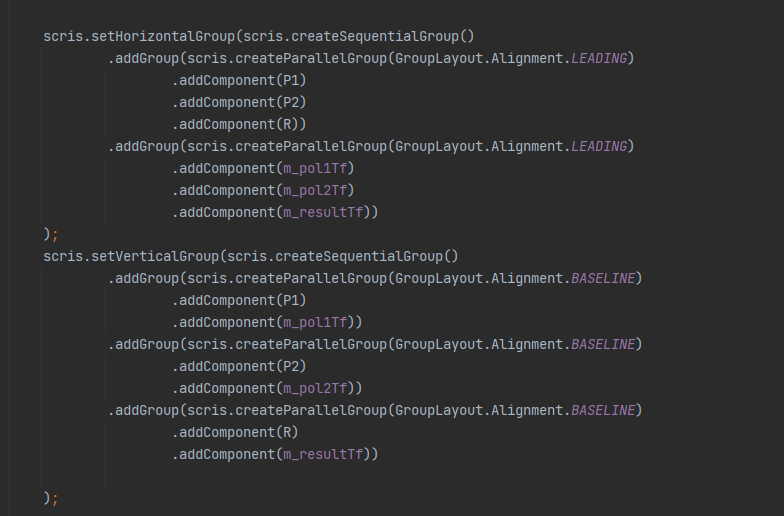
**View:** conține elementele care formează interfața grafică, precum butoanele sau text-field-urile, dar și câte un ActionListener pentru fiecare buton.

**Controller:** controlează logica aplicației, coordonând View-ul și Modelul. Acesta primește un input prin intermediul vederii, procesează datele cu ajutorul modelului si returnează rezultatul vederii.

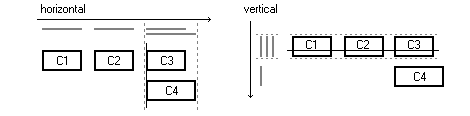
**4.2. GUI**

GUI (Graphical User Interface) din Java reprezintă un mecanism prin intermediul căruia utilizatorul poate interacționa cu un program. Acesta conține componente grafice, cum ar fi butoane, etichete, ferestre etc., pe care utilizatorul le poate utiliza pentru a interacționa cu ușurință cu aplicația. GUI are un rol important în construirea cu ușurință a unei interfețe pentru aplicațiile Java.

**View:** după cum am menționat în secțiunea anterioară, această clasă conține elementele care formează interfața grafică. Interfața văzută de utilizator este formată din trei panouri, JPanel din pachetul javax.swing. Panoul „poli” conține cele trei text-field-uri vizibile si etichetele cu textul „Polynomial 1”, „Polynomial 2”, respectiv „Result”. Doar text-field-urile corespunzătoare introducerii sunt editabile, pentru cel care afișează am dezactivat această opțiune pentru ca utilizatorul nu ar trebui sa poată modifica direct rezultatul.

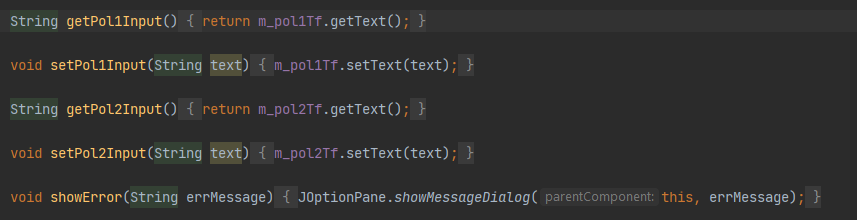
****Al doilea panou, „butoane” conține cele șapte butoane corepunzătoare opreațiilor de adunare, scădere, înmulțire, împărțire, integrare, derivare, respectiv ștergerea, sau revenirea la starea inițială, operație efectuată de butonul „Clear”. Layout-ul pentru aceste două panouri este de tip GroupLayout. În cazul utilizării acestui tip de layout componentele trebuie grupate atât pe orizontală, cât și pe verticală, motiv pentru care componentele sunt adăugate de două ori.

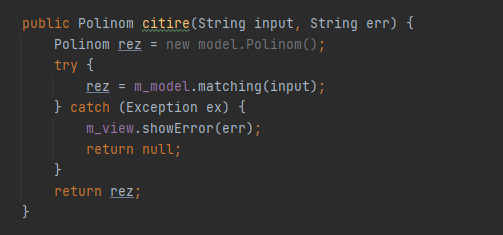
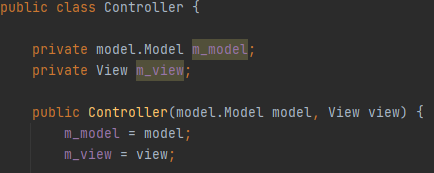
Acestea sunt adăugate în grupuri secvențiale și în grupuri paralele. Pentru a reuși să determin cu exactitate cum trebuie împărțite, imaginea următoare a fost de mare ajutor:

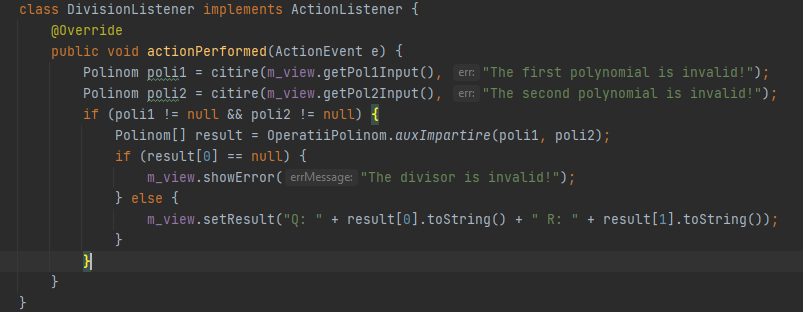
****

Cel de-al treilea panou și cel mai de bază, „panou” este cel care reprezintă aplicația propriu-zisă, în el fiind conținute celalalte două panouri.

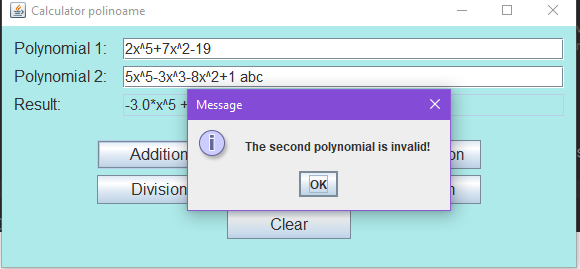
Tot în clasa View am creat metode de preluare a inputului, afișare a rezultatului sau a unei erori și ActionListener pentru fiecare buton.



**Controller:** conține logica aflată la baza interfeței grafice, motiv pentru care am ales să o menționez în acestă secțiune. Deoarece îmbina view-ul cu modelul, le conține ca atribute. În plus, pentru fiecare buton apelează metoda addOperațieListener, căreia îi este dat ca argument un nou obiect de tipul clasei OperațieListener. Astefl, clasa Controller încapsulează șapte clase: „ClearListener”, „AdditionListener”, „SubtractionListener”, „MultiplicationListener”, „DerivativeListener”, „IntegrationListener”, „DivisionListener”. Toate aceste suprascriu metoda „actionPerformed”, în funcție de caz citind polinomul introdus de utilizator, efectuând operația respectivă și aruncând o excepție de tipul Exception, atunci când datele introduse de utilizator nu sunt valide. În cazul clasei „ClearListener” pentru fiecare text-field este setat String-ul null. În acestă clasă am implementat și o metodă pentru citire. Polinomul rezultat primește valoarea returnată de funcția matching din Model, care aruncă o excepție dacă utilizatorul introduce date invalide. Mesajul de eroare este afișat cu ajutorul unui JOptionPane, apelând metoda showErr, cu mesajul corespunzător. De asemenea, metoda apelată pentru a efectua împarțirea este auxImpartire, pentru a trata cât mai multe dintre cazurile posibile care ar putea fi introduse de către utilizator.

****

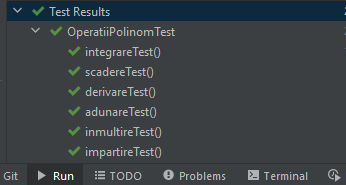
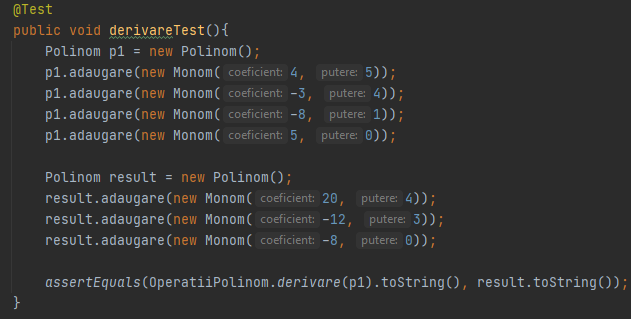
1. **Rezultate**
   1. **Funcționare**

În cazul operațiilor cu doi operanzi, utilizatorul va introduce polinomul dorit în text-field-urile corespunzătoare fiecăruia dintre ele. În momentul în care apasă pe unul dintre butoane pentru a efectua operațiile, dacă input-ul este invalid va apărea un dialog box care va preciza care dintre polinoame este greșit. Dacă datele introduse sunt corecte, se va efectua operația și rezultatul va fi afișat in câmpul corespunzător.

Iar, în cazul operațiilor cu un singur operand, i.e. integrarea și derivarea doar primul polinom este luat în considerare. Input-ul introdus în câmpul „Polynomial 2” nu va fi luat in considerare.

* 1. **Testare cu JUnit**

****Pentru testarea cu JUnit am creat, în pachetul test, o nouă clasă, „OperatiiPolinomTest”. În interiorul acesteia, am creat câte o metodă de testare pentru fiecare operație. În interiorul acestor metode, am creat trei polinoame, două dintre acestea vor fi folosite ca operanzi, iar cel de-al treilea reprezintă rezultatul așteptat în urma efectuării operației respective. Ulterior, cu ajutorul lui assertEquals, am testat egalitatea dintre polinomul rezultat si cel la care ne-am aștepta să il primim ca rezultat. Spre exemplu, în cazul opeației de înmulțire am polinoamele 4x^5-3x^4-8x+5 și 4x^3+12, a căror rezultat așteptat este cel din imaginea alăturată.

****Pentru metodele cu un singur operand, de exemplu derivarea, am creat doar doua polinoame, cel care urmează să fie derivat si rezultatul așteptat. Și în acest caz mă folosesc de assertEquals pentru a testa egalitatea dintre rezultatul actual și cel așteptat.

Deoarece a trecut testele pentru cazurile pe care le-am dat în JUnit și pe parcursul implementării, am considerat că acest calculator de polinoame calculează corect.

1. **Concluzii**

Datorită acestui proiect am învățat să folosesc atât regex, pentru a testa dacă un input respectă un anumit tipar, cât și testarea cu JUnit. Cu ajutorul acestuia voi putea testa mai ușor funcționarea corectă a programelor viitoare.

Cu toate acestea, unul dintre lucrurile pe care aș dori să le îmbunătățesc, legate de acest proiect, este regex-ul, aș dori ca pattern-ul să nu fie atât de restrictiv, relativ la ce este considerat ca fiind un polinom. Însă pentru a face acest lucru consider că aș avea nevoie de mai multa experiență.

1. **Bibliografie**

* Definiție GUI (17.03.2021 21:30): [https://www.guru99.com/java-swing-gui.html definiție GUI 17.03](https://www.guru99.com/java-swing-gui.html%20definiție%20GUI%2017.03)
* Layout (7.03.18:00): <https://docs.oracle.com/javase/tutorial/uiswing/layout/group.html>
* Împărțire polinoame (15.03.2021 19:00): <https://en.wikipedia.org/wiki/Polynomial_long_division>
* Regex (9.03.2021 20:00): <https://crunchify.com/what-is-regex-pattern-regular-expression-how-to-use-it-in-java-example-attached/>