DOCUMENTAȚIE

TEMA 1 – *Calculatorul Polinomial*

Ciochină Cătălina- Andreea

Facultatea de Calculatoare și Tehnologia Informației

Grupa 30228

Prof. coordonator,

Chifu Viorica Rozina

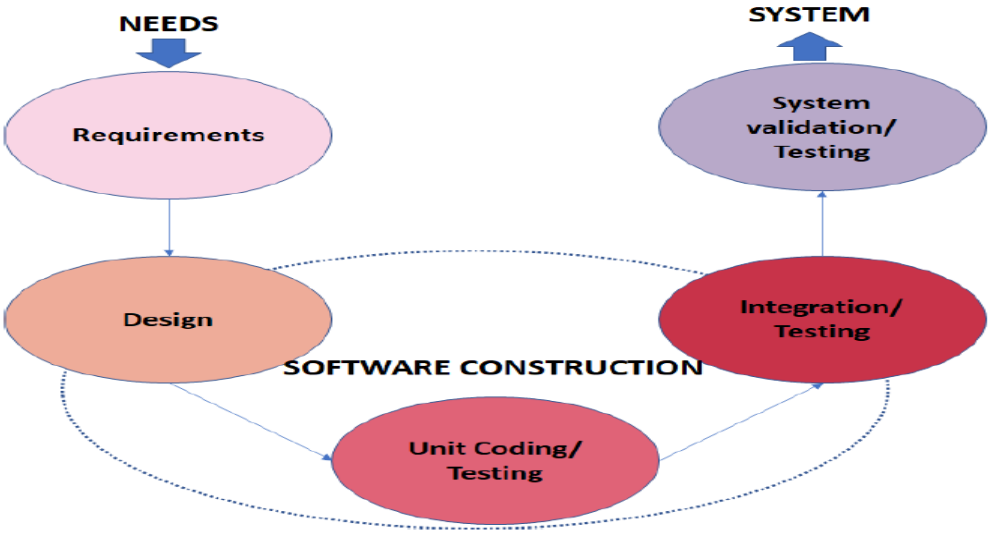
NUME STUDENT: Ciochină Cătălina-Andreea
GRUPA: 30228

CUPRINS

1. Obiectivul temei

Obiectivul principal al calculatorului polinomial este de a oferi utilizatorilor săi uneltele necesare pentru a efectua operații matematice complexe cu polinoame, precum adunarea, scăderea, înmulțirea, împărțirea, derivarea și integrarea, care în cazul unor polinoame de grad mare, ajung să fie destul de greu de realizat pe foaie, greșelile strecurându-se foarte ușor.

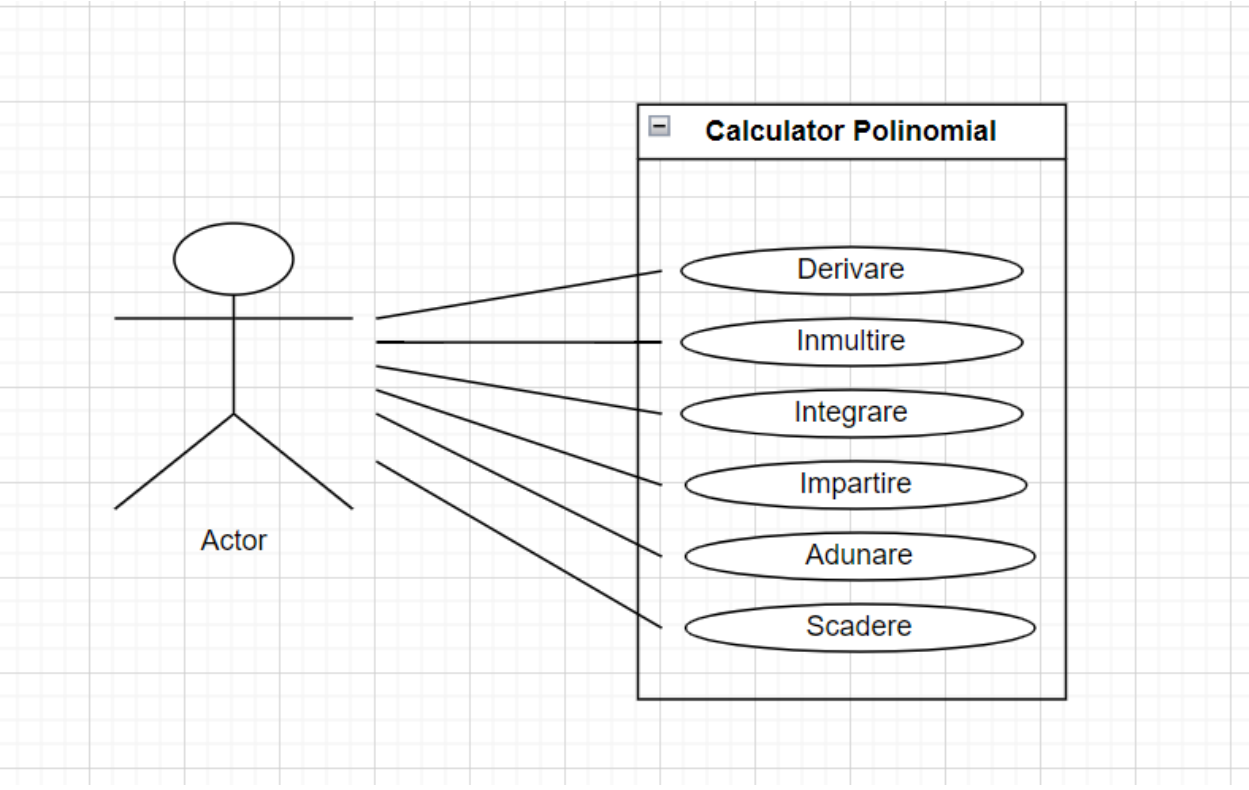
Obiective secundare	***Detalii***	***Secțiunea***
Colectarea și analiza cerințelor problemei	***Colectarea*** și ***analiza*** cerințelor problemei reprezintă procesul de înțelegere a nevoilor și cerințelor utilizatorului și transformarea acestora în specificații clare pentru dezvoltarea software-ului.	2
Proiectarea soluției	După ce cerințele au fost analizate, trebuie să se ***proiecteze*** o soluție software care îndeplinește cerințele utilizatorului.	3
Implementarea soluției	***Scriere de cod*** pentru dezvoltarea unui sistem software care să corespundă proiectării și analizei realizate anterior.	4
Testarea	Se face cu scopul de a întregii produsul software și de a ne asigura că implementarea este corectă pentru ***toate cazurile posibile***, pe care user-ul le-ar putea introduce/genera.	5



2. Analiza problemei, modelare, scenarii, cazuri de utilizare

2.1. Cerințe functionale

- Sistemul trebuie să permită utilizatorului să introducă 2 polinoame.
- Sistemul trebuie să permită utilizatorului selectarea operației și schimbarea acesteia pe polinoamele deja introduse.
- Sistemul trebuie sa extragă coeficientul și gradul pentru fiecare termen al polinomului introdus sub forma de șir de caractere de user prin intermediul interfeței.
- Efectuarea operațiilor asupra celor două polinoame introduse și parsate (adunare, scădere, înmulțire, împărțire, derivare, integrare).
- Sistemul trebuie să permită repetarea procesului.



2.2. Cerințe non-funcționale

1. *Performanța*: calculatorul trebuie să fie capabil să efectueze calcule complexe și să furnizeze rezultate într-un timp rezonabil.
2. *Ușurința în utilizare*: interfața grafică trebuie să fie ușor de folosit pentru utilizatorii finali.
3. *Extensibilitate*: calculatorul trebuie să permită adăugarea de noi funcționalități și opțiuni pentru a satisface nevoile viitoare ale utilizatorilor.
4. Scalabilitate: sistemul trebuie să fie capabil să facă față unei cantități mari de date și să poată gestiona sarcini multiple simultan.
5. *Reutilizabilitate*: codul trebuie să fie modular și ușor de reutilizat pentru a putea fi folosit în alte proiecte viitoare.
6. *Fiabilitate*: calculatorul trebuie să funcționeze fără erori și să ofere rezultate precise.
7. *Securitate*: datele introduse în calculator și rezultatele generate trebuie să fie protejate și accesibile numai utilizatorilor autorizați.
8. *Portabilitate*: sistemul trebuie să poată fi instalat și utilizat pe mai multe platforme hardware și software.

2.3. Scenarii de utilizare/ Use cases

Pentru toate operațiile este același mod de operare:

Use Case: Operație (Adunare, Scădere, Înmulțire, Împărțire, Derivare, Integrare)

Primary Actor: User-ul

Main Success Scenario:

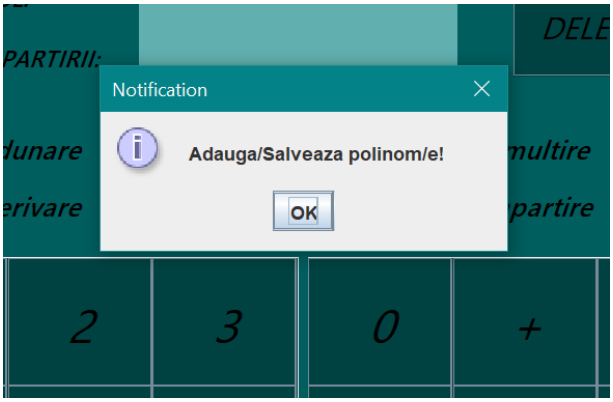
1. User-ul introduce primul polinom și apasă butonul de SAVE, asociat primului polinom, în cazul în care operația pe care o dorește se realizează pe două polinoame, user-ul introduce și cel de-al doilea polinom și apasă butonul SAVE, asociat celui de-al doilea polinom.

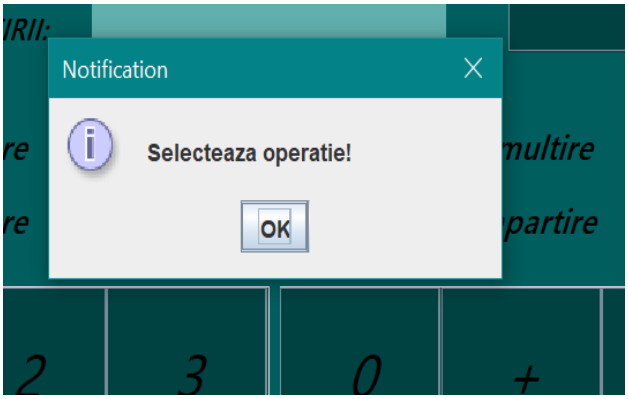
2. User-ul selectează operația pe care o dorește să se realizeze pe polinoamele introduse.

3. User-ul apasă butonul de EGAL și rezultatul operației dorite va apărea in cadrul casetei de text Rezultat (+Rest, în cazul împărțirii).

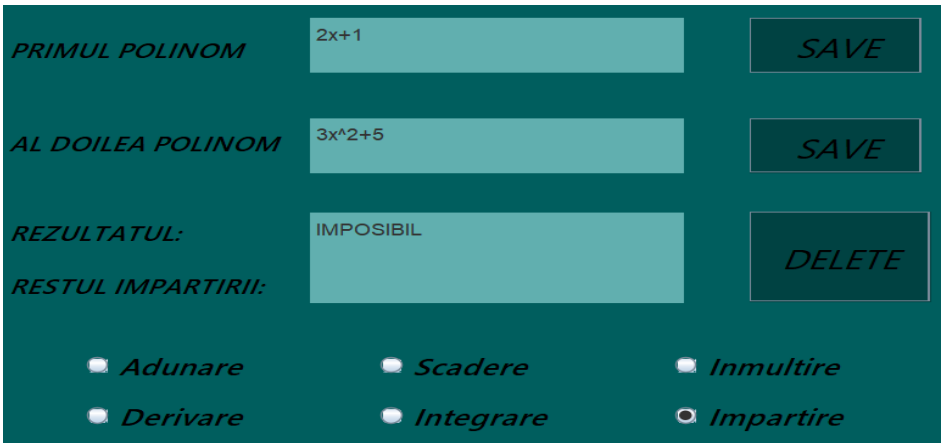
Alternative Sequence: Polinoame introduse incorrect

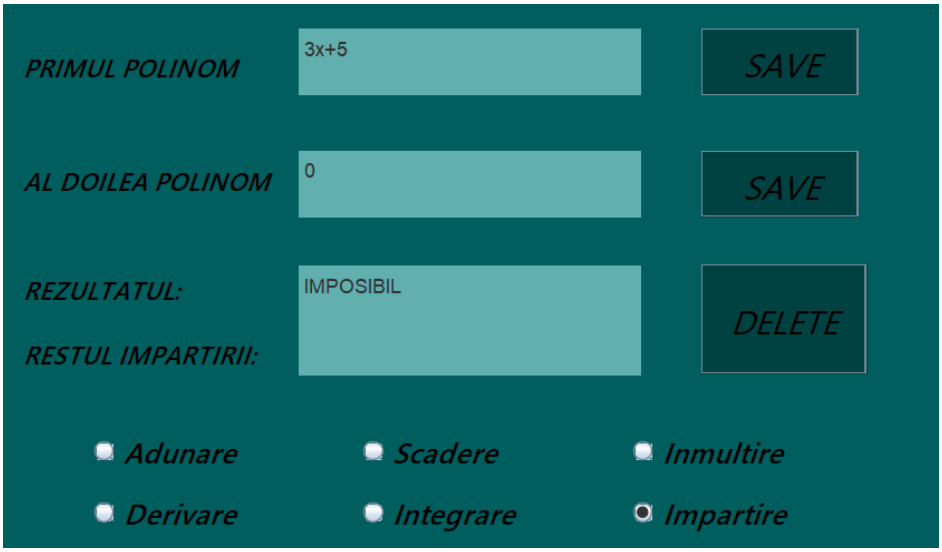
Vor apărea pop-up-uri pentru cazurile când polinoamele nu au fost introduse sau au fost introduse greșit.





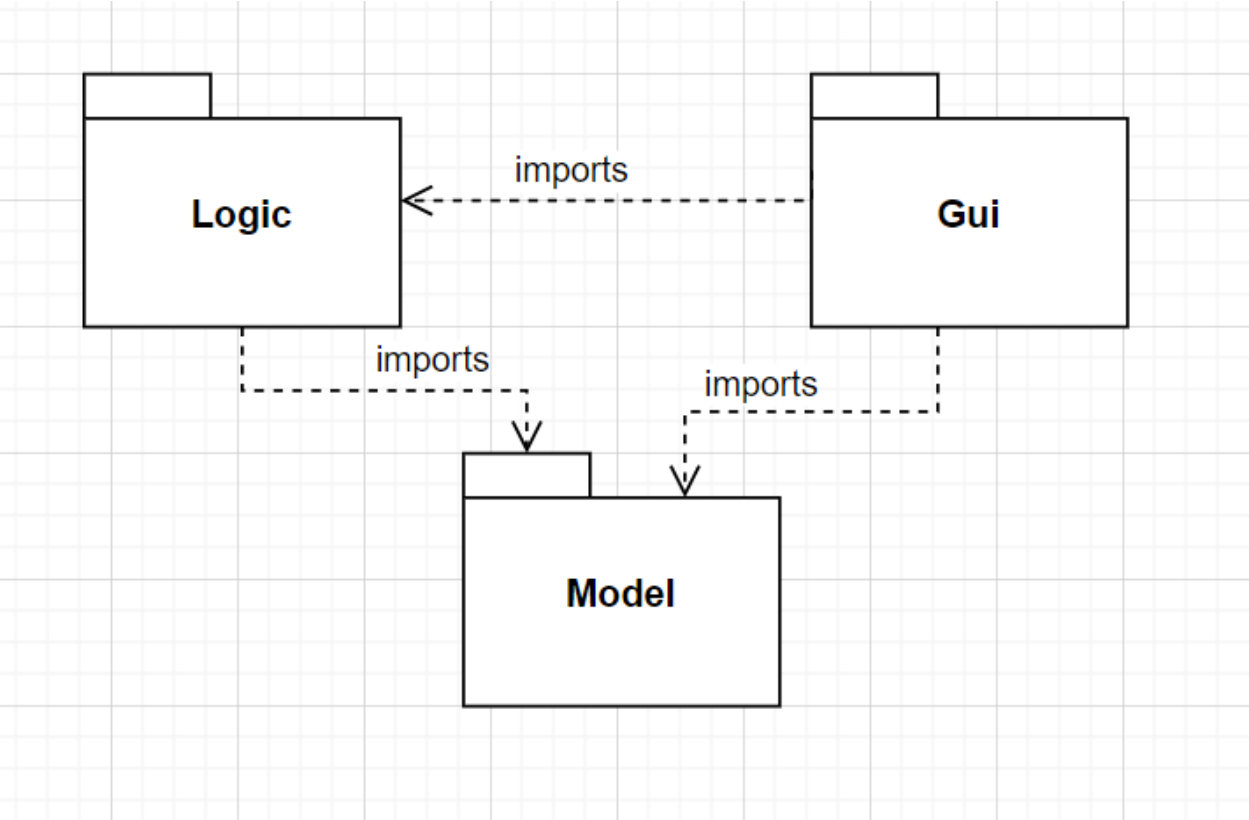
În cazul împărțirii, atunci când primul polinom introdus are gradul mai mic decât celălalt, în căsuța de text va apărea IMPOSIBIL, întrucât operația cerută nu este posibilă. Asemenea se întamplă și în cazul introducerii unui polinom 0 (la împărțirea la 0).





3. Proiectare

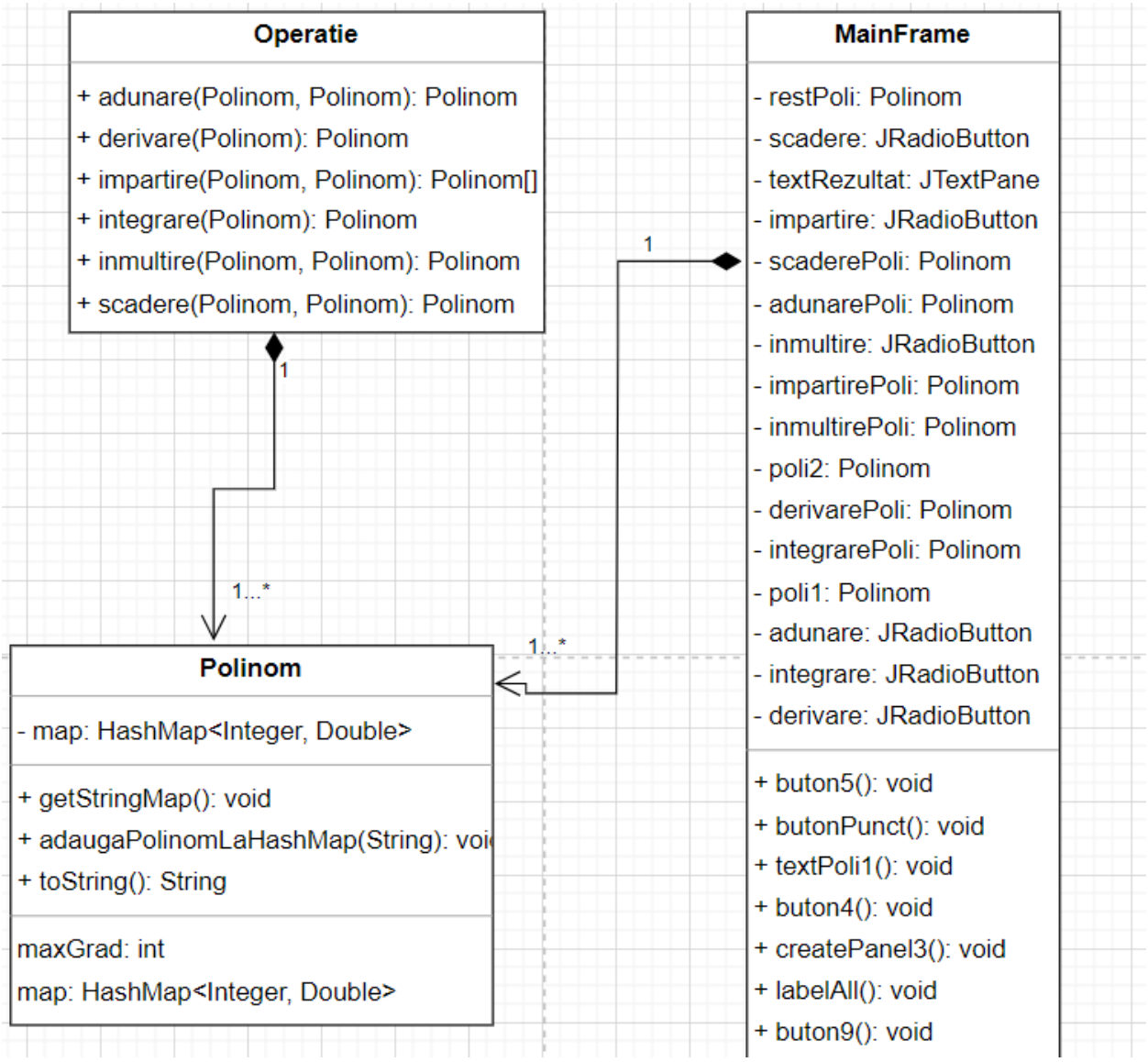
Am organizat codul în pachete asemenea organizării Model-View-Controller, denumirea în cadrul proiectului meu este Model(cu clasa Polinom)-Logic(cu clasa Operație)-GUI(cu clasa MainFrame și Controller).



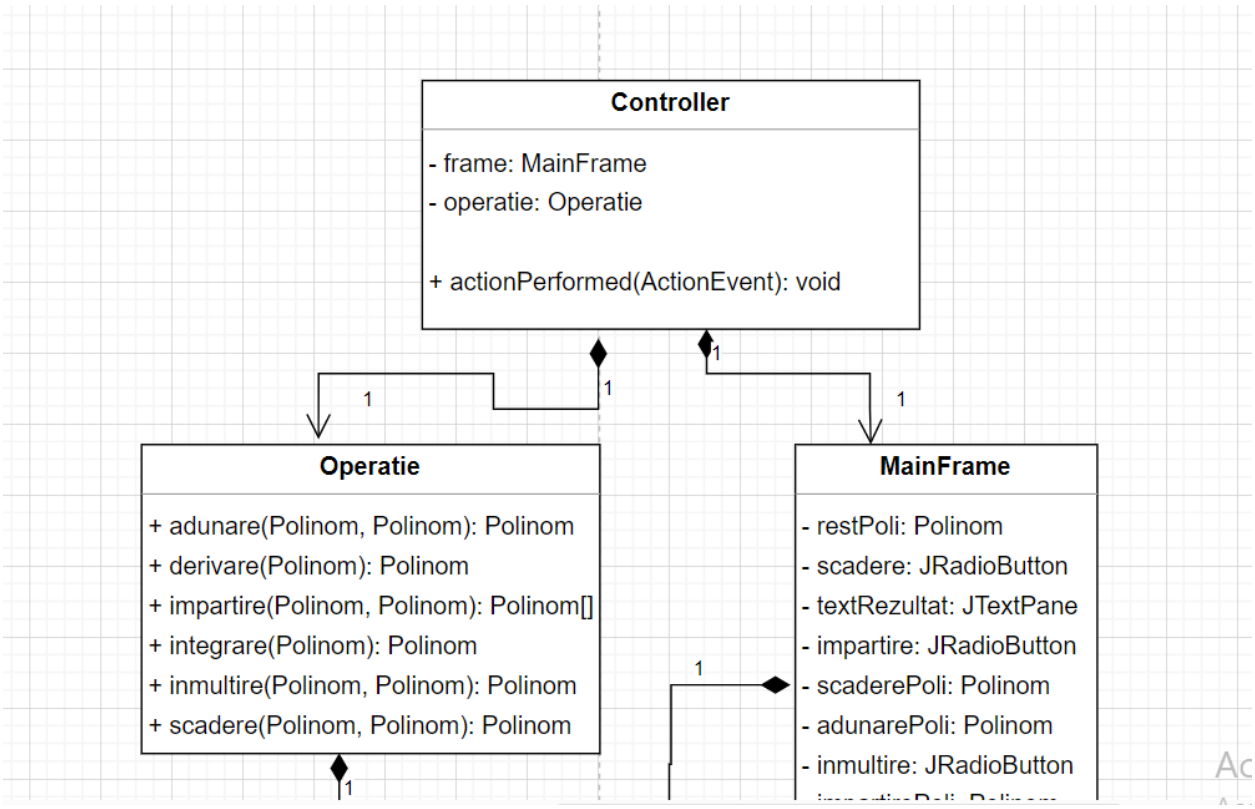
Clasele pe care le-am creat sunt descrise în următoarea secțiune(4. Implementare)

1. Clasa Polinom - MODEL
2. Clasa Operatie - LOGIC
3. Clasa Controller - GUI
4. Clasa MainFrame - GUI
5. Clasa OperatiiParamTest - TEST

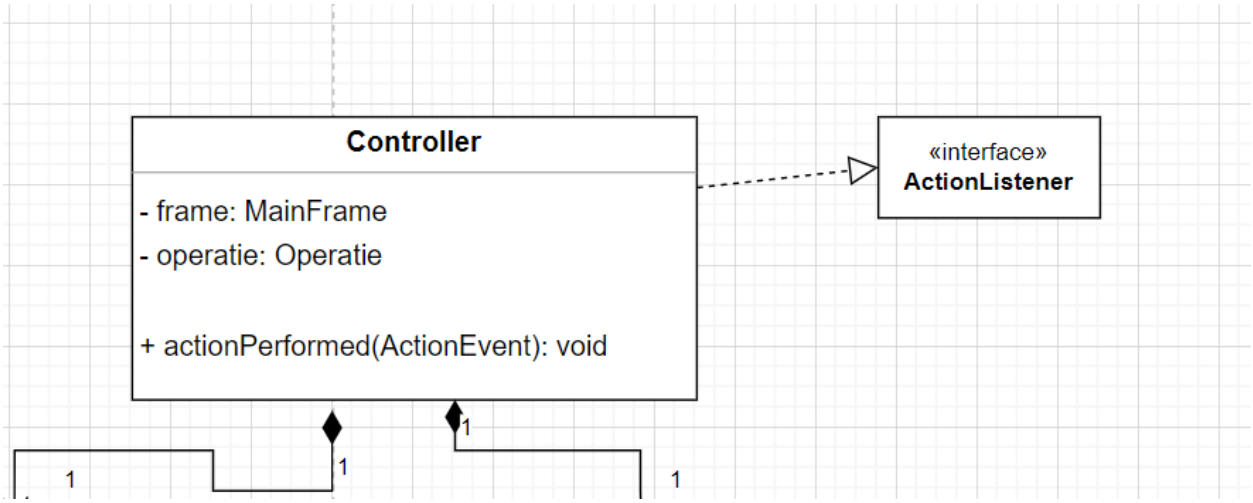
Diagrama UML de clase corespunzătoare implementării pe care am făcut-o este următoarea, am despărțit-o în mai multe poze din cauza numeroaselor metode și atribute ale MainFrame:



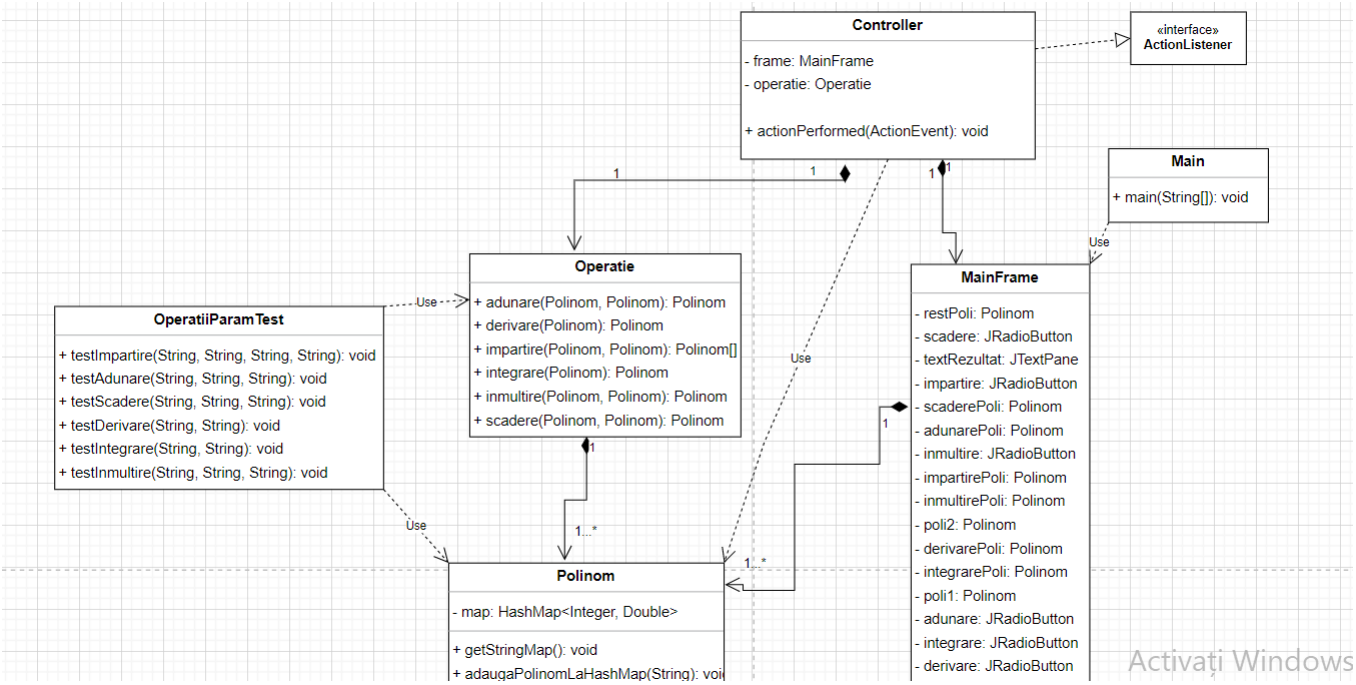
Relație de compoziție, 1 sau mai multe polinoame sunt parte a unei Operații sau a unui MainFrame.



Un Controller are o Operatie si un MainFrame(sunt parte a unui Controller), implementează interfața ActionListener și folosește clasa Polinom.



Clasa de teste foloseste clasa Operatie si Polinom și Main pe MainFrame ca să creeze o fereastră de interfață.



4. Implementare

❖ *Clasa Polinom*

Este responsabilă de reprezentarea interna a polinoamelor introduse de user. Are o variabilă instanță de tipul HashMap, care este alocată în constructor, cu valori nule pentru fiecare obiect nou creat de tipul clasei. Pentru aceasta variabilă avem setter și getter.

În cadrul acestei clase avem și o metodă care primește un String prin care adăugăm la HashMap coeficienții și gradele unui polinom(string-ul care vine de la user). Metoda creează un obiect de tipul Pattern și metoda compile compilează o expresie regulată și o transformă într-un model de expresie regulată utilizat pentru a găsi potriviri într-un șir de caractere. Aceasta utilizează caractere speciale pentru a defini tiparele, cum ar fi \\d pentru orice cifră, \\. pentru punctul zecimal, \\+ și \\- pentru semne aritmetice, x pentru variabila x și \\^ pentru exponent. Metoda matcher() a obiectului Pattern returnează un obiect Matcher care poate fi utilizat pentru a căuta potriviri în șirul de caractere polinom. Instrucțiunile din interiorul blocului while parcurg toate potrivirile găsite de obiectul Matcher, extrag coeficientul și gradul din fiecare termen și îl adaugă într-un HashMap numit map.

Instrucțiunile din interiorul blocului while parcurg toate potrivirile găsite de obiectul Matcher, extrag coeficientul și gradul din fiecare termen și îl adaugă în HashMap polinomului. În primul rând, se declară și se inițializează variabilele grad, coef și semn. Variabila termen primește valoarea următoarei potriviri găsite de Matcher. Dacă termenul începe cu semnul minus, semnul este setat la -1.0, iar termenul este modificat pentru a elimina semnul minus. Dacă termenul începe cu semnul plus, termenul este modificat pentru a elimina semnul plus.

Se verifica apoi tipul termenului și se extrage coeficientul și gradul termenului utilizând expresii regulate. Dacă termenul este de forma ax^n, unde a și n sunt numere întregi sau zecimale, se separă coeficientul și gradul din termen și se atribuie variabilelor coef și grad. Dacă termenul este de forma ax, se extrage doar coeficientul și se setează gradul la 1. Dacă termenul este de forma x^n, se extrage gradul și se setează coeficientul la 1. Dacă termenul este de forma x, se setează coeficientul la 1 și gradul la 1. Dacă termenul este de forma a, unde a este un număr întreg sau zecimal, se extrage coeficientul și se setează gradul la 0.

În cele din urmă, coeficientul termenului este adăugat în HashMap la cheia corespunzătoare gradului, utilizând metoda put(). Dacă cheia gradului nu există în HashMap, se adaugă o nouă pereche cheie-valoare în HashMap, iar valoarea este setată la coeficientul termenului. Dacă cheia gradului există deja în HashMap, valoarea asociată acestei chei este actualizată prin adăugarea coeficientului termenului curent la valoarea existentă, utilizând metoda getOrDefault(), în cazul în care user-ul introduce 2 sau mai multe elemente de același grad.

O altă metodă a acestei clase Polinom este getMaxGrad() prin care se obține gradul maxim din HashMap-ul acelui obiect creat. Metoda getStringMap() a fost utilă înainte de implementarea interfeței, afișează în consolă gradele și coeficienții din HashMap.

O altă metodă importantă este toString(), pe care am suprascris-o astfel încât HashMap-ul să fie transformat într-un string user-friendly, pentru interfață.

❖ ***Clasa Operatie***

Conține 6 metode care implementează operațiile asupra polinoamelor:

- ***Adunarea***

Punem valorile primului polinom într-un nou polinom și parcurgem valorile din poli2, se adună coeficienții pentru gradele comune celor două și se pun neschimbați cei necomuni.

```
public Polinom adunare(Polinom poli1,Polinom poli2) {
    Integer exp;
    Polinom adun = new Polinom();
    adun.getMap().putAll(poli1.getMap());

    for (Map.Entry<Integer, Double> entry : poli2.getMap().entrySet()){
        exp = entry.getKey();
        if( adun.getMap().get(exp)!=null)
            adun.getMap().replace(exp,adun.getMap().get(exp)+poli2.getMap().get(exp));
        else
            adun.getMap().put(exp,poli2.getMap().get(exp));
    }
    return adun;
}
```

- ***Scăderea***

Punem valorile primului polinom într-un nou polinom și parcurgem valorile din poli2, se scad coeficienții pentru gradele comune celor două și se pun cu semn schimbat cei necomuni (*(-1)).

```
public Polinom scadere(Polinom poli1,Polinom poli2) {
    Integer exp;
    Polinom scad = new Polinom();
    scad.getMap().putAll(poli1.getMap());

    for (Map.Entry<Integer, Double> entry : poli2.getMap().entrySet()){
        exp = entry.getKey();
        if(scad.getMap().get(exp)!=null)
           scad.getMap().replace(exp,scad.getMap().get(exp)-poli2.getMap().get(exp));
        else
            scad.getMap().put(exp,-poli2.getMap().get(exp));
    }
    return scad;
}
```

- ***Înmulțirea***

Înmulțirea a două polinoame presupune ca fiecare element dintr-un polinom să fie înmulțit cu fiecare element din celălalt element. Astfel, parcurgem cele două polinoame deodată și exp = i+j, dacă exponentul exista adunăm și coeficientul obținut la acest pas, iar dacă nu există punem coeficientul.

```
public Polinom inmultire(Polinom poli1,Polinom poli2) {
    Polinom multi = new Polinom();
    for (Integer i  : poli1.getMap().keySet()) {
        for (Integer j  : poli2.getMap().keySet()) {

            Integer exp = i+j;
            Double coef = poli1.getMap().get(i)* poli2.getMap().get(j);
            if (multi.getMap().containsKey(exp)) {
                multi.getMap().replace(exp, multi.getMap().get(exp) + coef);
            } else {
                multi.getMap().put(exp, coef);
            }
        }
    }
    return multi;
}
```

- ***Împărțirea***

Metoda de împărțire trebuie să returneze un vector de Polinoame, 2 mai exact, rezultatul și restul împărțirii. Pentru fiecare polinom stabilim din start gradul maxim și dacă polinomul 2 are gradul maxim mai mare decât polinomul1 atunci operația nu este posibilă.

```
public Polinom[] impartire(Polinom poli1, Polinom poli2) {
    Polinom[] rezultat = new Polinom[2];
    Polinom poliImpartire = new Polinom();
    Polinom poliRest = new Polinom();
    Polinom pol1 = new Polinom();pol1.getMap().putAll(poli1.getMap());
    Polinom pol2 = new Polinom();pol2.getMap().putAll(poli2.getMap());
    Polinom produs = new Polinom();

    int gradMax1 = pol1.getMaxGrad();
    int gradMax2 = pol2.getMaxGrad();
    if (gradMax1 < gradMax2 || gradMax2 == 0 || gradMax1 == 0) {
        rezultat[0] = poliImpartire;
        rezultat[1] = poliRest;
        return rezultat;
    }
```

Cât timp gradul maxim al primului este mai mare decât gradul celui de-al doilea polinom și sunt diferite de 0, atunci împărțim cei doi coeficienți(pentru exp cu grad maxim) si punem în polinomul de împarțire coeficientul la diferența exponenților. Apoi inmultim fiecare elememt din polinomul de impartire cu polinomul 2 si dupa le scadem din polinomul 1 si repetam pașii.

```
    while(gradMax1>=gradMax2 && gradMax2!=0 && gradMax1!=0){
        produs.getMap().clear();
        double coef = pol1.getMap().get(gradMax1) /pol2.getMap().get(gradMax2);
        poliImpartire.getMap().put(gradMax1 - gradMax2,coef);

        for (Integer grad : pol2.getMap().keySet()) {
            double coef2 = coef * pol2.getMap().get(grad);
            produs.getMap().put(grad + gradMax1 - gradMax2, coef2);
        }
```

Ce ne rămâne la final, atunci când iesim din while, în polinomul 1 este chiar polinomul de rest.

```
        for (Integer degree : produs.getMap().keySet()) {
            if (pol1.getMap().containsKey(degree)) {
                double coeff = pol1.getMap().get(degree) - produs.getMap().get(degree);
                if (coeff == 0)
                    pol1.getMap().remove(degree);
                else
                    pol1.getMap().put(degree, coeff);
            }
            else
                pol1.getMap().put(degree, -produs.getMap().get(degree));
        }
        gradMax1 = pol1.getMaxGrad();
    }

    rezultat[0] = poliImpartire;
    rezultat[1] = pol1;
    return rezultat;
}
```

- ***Integrare***

Integrarea la polinoame presupune ca pentru fiecare element din polinom, creșterea exponentului exp++ și coeficientul să ia valoarea coeficientului actual / valoarea exponentului. Adăugarea constantei C am făcut-o direct în caseta de text din interfață.

```
public Polinom integrare(Polinom poli){
    Polinom poliIntegrat = new Polinom();
    for(Integer i : poli.getMap().keySet()){
        Integer exp = i; exp++;
        Double coef = poli.getMap().get(i);
        Double coef2 =coef * (1.0/exp);

        DecimalFormat form = new DecimalFormat( pattern: "#.###",new DecimalFormatSymbols(Locale.US));
        String formatted = form.format(coef2);
        coef2 = Double.parseDouble(formatted);

        poliIntegrat.getMap().put(exp,coef2);
    }
    return poliIntegrat;
}
```

- ***Derivare***

Derivarea la polinoame presupune ca pentru fiecare element din polinom, coeficientul să ia valoarea exponent*coeficient actual și exp--, în cazul în care exp>0.

```
public Polinom derivare(Polinom poli){
    Polinom poliDerivat = new Polinom();
    for(Integer i : poli.getMap().keySet()){
        Integer exp = i;
        if(exp>0)
            exp--;
        Double coef = poli.getMap().get(i);
        Double coef2 = i * coef;
        poliDerivat.getMap().put(exp,coef2);
    }
    return poliDerivat;
}
```

❖ *Clasa MainFrame*

Clasa MainFrame extinde clasa JFrame și împreună cu clasa Controller au drept prim scop implementarea unei interfețe user-friendly**.**

Clasa MainFrame, are câte o metodă pentru fiecare buton din interfață. Implicit aplicația începe prin scrierea polinomului 1 în caseta text și apoi a celui de-al doilea, însă focus-ul se poate schimba prin apăsarea pe caseta dorită.

După introducerea polinomului 1 și apăsarea butonului de SAVE, focus-ul este transferat urmatoarei casete de text ce reprezintă cel de-al doilea polinom. De asemenea și salvarea acestui polinom se face tot prin apăsarea butonului de SAVE. Pentru butoanele de cifre,x,-,+,.,^ în funcție de care casetă de text are focus=1, se scrie simbolul respectiv. Acestea sunt grupate în 2 panel-uri.

Inițializarea ferestrei de aplicație se face tot printr-o metodă, am stabilit ca închiderea aplicației să se facă la apăsarea ieșirii X din colt dreapta sus. Tot aici adăugăm butoanele de SAVE, butonul de DELETE care la apăsarea sa produce un refresh, mai exact șterge toate datele anterioare și permite user-ului să reintroducă polinoamele, butonul INAPOI care cand este apăsat șterge cu câte un caracter din caseta care are focus în acel moment, iar butonul STERGE, elimină tot string-ul din caseta care are focus.

Cel de-al treilea panel conține un grup de butoane radio care selectează operația care se dorește a fi făcută asupra polinoamelor.

❖ *Clasa Controller*

Butonul de EGAL din MainFrame are adaugat drept ActionListener un obiect de tipul clasei Controller, care implementează interfața ActionListener. În cadrul MainFrame, la butonul EGAL am adăugat și comanda "Egal", ca să identificăm în Controller pentru care acțiune trebuie să reacționeze.

Clasa Controller are 2 variabile instanță, una de tipul Operatie și una de tipul Mainframe. Trebuie implementată metoda interfeței actionPerfomed(ActionEvent e). Aici dacă comanda este Egal, atunci se trece prin mai multe întrebări (s-au completat polinoamele, s-a selectat operația, care operație este). În funcție de care operație este bifată, în caseta Rezultat(+Rest, în cazul împărțirii) se stabilește polinomul rezultat al cărei operații să apară. Aici am stabilit și greșelile care pot să apară, un fel de evitare a excepțiilor, în cazul în care polinoamele nu au fost completate sau nu au fost salvate va apărea un pop-up care îi comunică utilizatorului că nu a introdus polinoamele.

❖ *Clasa OperatiiParamTest*

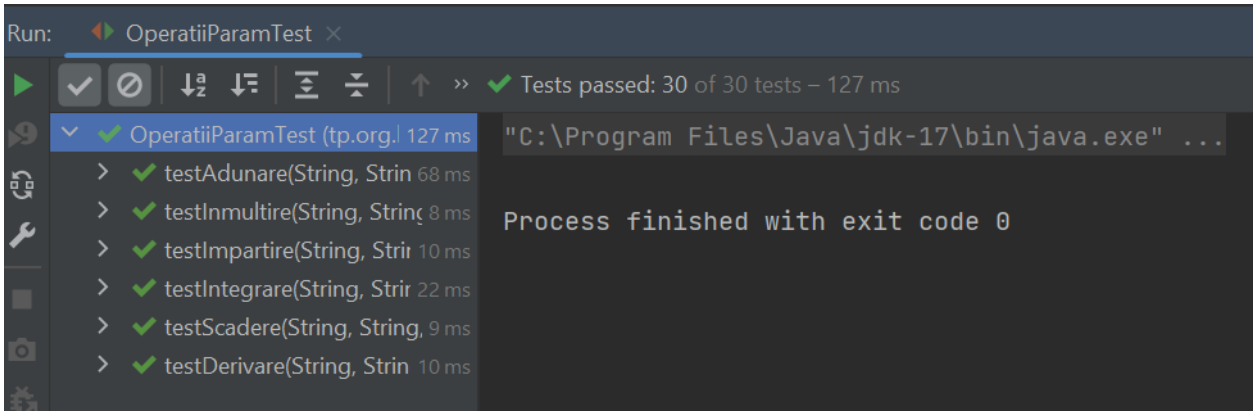
În această clasă am făcut testarea operațiilor, am folosit Framework-ul Junit. Pentru fiecare operație cu ajutorul testării parametrizate

5. Rezultate

Cu ajutorul testării parametrizate am făcut mai multe teste pentru fiecare operație asupra polinoamelor, acestea finalizându-se cu success(30 de teste). Am încercat să aplic toate cazurile posibile și chiar am introdus valori obiective, pe care le-am obținut de pe diverse site-uri de matematică de liceu, astfel încât să evit vreo eroare personală de concept a vreunei operații.

```
@ParameterizedTest
@CsvSource({"x^2-2x+1,x-1,1.0x-1.0,0.0",
            "x^2+2x+1,x+1,1.0x+1.0,0.0",
            "x^2+x,0,0.0,0.0",
            "2x^4+x^2+x-3,x^2-4x,2.0x^2+8.0x+33.0,133.0x-3.0",
            "x^3+3x^2-x-3,x-2,x^2+5.0x+9.0,15.0"})
public void testImpartire(String poli1,String poli2,String rezultatImp,String rezultatRest) {
    Operatie operatie = new Operatie();
    Polinom polinom1 = new Polinom();
    polinom1.adaugaPolinomLaHashMap(poli1);
    Polinom polinom2 = new Polinom();
    polinom2.adaugaPolinomLaHashMap(poli2);
    Polinom[] poli =  new Polinom[2];
    poli = operatie.impartire(polinom1,polinom2);
    assertEquals(rezultatImp,poli[0].toString());
    assertEquals(rezultatRest,poli[1].toString());
}
```

Testarea a avut loc cu succes pentru fiecare operație, după cum urmează:



6. Concluzii

Consider că acest proiect m-a ajutat să îmi aprofundez cunoștințele acumuluate semestrul trecut în cadrul cursului de Programare Orientata pe Obiect. De asemenea, am învățat să lucrez cu Framework-ul Junit și cu expresii regulate, care sunt tehnologii extraordinar de utile și captivante. Pot să spun și că am învățat că este mult mai corect să îmi aranjez clasele in pachete, care să urmeze o arhitectură MVC. Mi-am dezvoltat și abilitățile de proiectare software în Java, astfel încât să urmeze principiile OOP cât mai bine.

Ca o dezvoltare viitoare, cred că ar fi interesant să se introducă și alte operații precum aflarea rădăcinilor polinomului sau a punctelor de extrem, poate și reprezentări grafice, tabele de existență.

7. Bibliografie

https://dsrl.eu/courses/pt/
https://users.utcluj.ro/~igiosan/teaching_poo.html
https://www.guru99.com/junit-parameterized-test.html
https://topic.alibabacloud.com/a/use-a-regular-expression-regex-to-match-polynomials-polynomial-regexpolynomial_1_31_32668196.html
https://www.draw.io/
https://docs.oracle.com/javase/tutorial/essential/regex/
https://www.javatpoint.com/java-regex