

DOCUMENTATIE

TEMA *1*

NUME STUDENT:... Kiraly Nicole Elena...
GRUPA: ...30227.....

CUPRINS

1.	Obiectivul temei.....	3
2.	Analiza problemei, modelare, scenarii, cazuri de utilizare	3
3.	Proiectare	5
4.	Implementare	7
5.	Rezultate	13
6.	Concluzii.....	17
7.	Bibliografie	18

1. Obiectivul temei

Obiectivul principal al temei 1 este implementarea unui calculator polinomial, care poate executa una dintre operatiile urmatoare: adunare, scadere, inmultire, impartire, derivare sau integrare.

Obiectivele secundare pentru indeplinirea obiectivului principal sunt:

- Realizarea interfetei grafice in clasa View(GUI)
- Realizarea claselor necesare pentru implementare – Monomial, Polinomial (Data Models), Operations(BusinessLogic), Controller(GUI), Main(GUI)
- Folosirea notiunilor programarii orientate pe obiecte
- Implementarea operatiilor-adunare, scadere, inmultire, impartire, derivare si integrare
- Folosirea expresiilor regulate si a pattern-ului si matcher-ului pentru extragerea coeficientilor si puterilor fiecarui monom ce alcatuiesc polinomul
- Testarea unitara a operatiilor pentru verificarea corectitudinii utilizand JUnit

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

Pentru rezolvarea temei, trebuie sa analizam urmatoorii pasi principali:

- Definirea problemei
- Intelegerea problemei si domeniul sau
- Analiza problemei
- Gasirea solutiilor pentru sub-probleme individuale
- Construirea solutiei finale

In cazul calculatorului polinomial, este necesar sa intelegem ce este un monom, ce este un polinom, care sunt operatiile ce se pot aplica pe unul sau mai multe polinoame, precum si cum se poate utiliza interfata grafica in concordanta cu acestea.

Pentru inceput, se construiesc clasele Monomial si Polinomial, apoi se incepe implementarea operatiilor care se pot aplica pe polinoame. Trebuie sa fie luate in considerare toate cazurile posibile pentru realizarea cat mai corecta a calculatorului. Deci, cazurile in care apar mai multi de x sau + sau – trebuie tratate.

Aplicatia este impartita in pachete: GUI, DataModels, BusinessLogic si Test.

Polinomul este format din monoame, care sunt scrise sub forma “coeficientx^putere”, sau “coeficient”, sau “coeficientx”, iar polinomul se reprezinta ca un Hashmap din monoame.

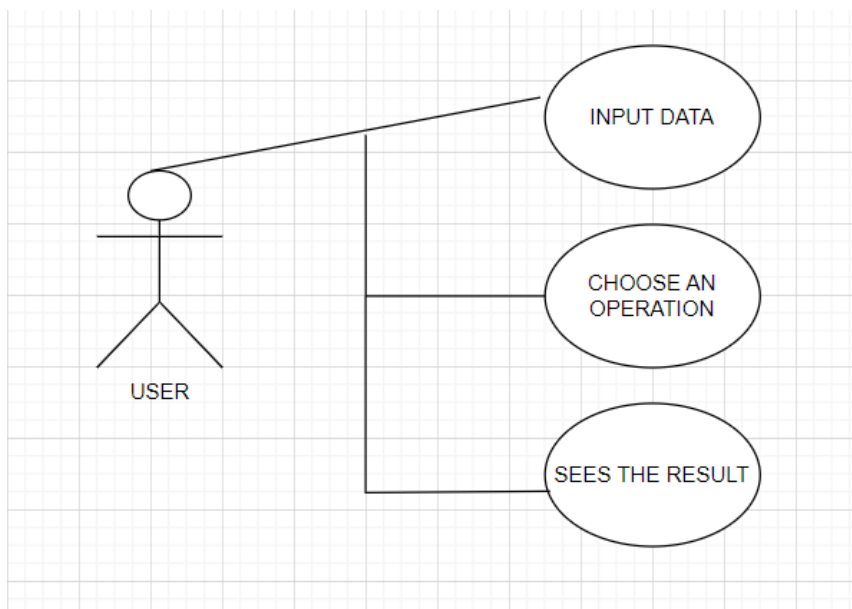
Dupa rularea aplicatiei, se deschide interfata unde introducem un polinom sau doua in functie de ce operatie vrem sa realizam. De exemplu, in JTextField-ul corespunzator primului polinom introducem un polinom sub forma “ $3x^2+4x^1+5x^0$ ”,sau sub forma “ $3x^2+4x+5$ ”, apoi introducem al doilea polinom respectand una dintre formele mentionate. Apasam butonul corespunzator operatiei pe care vrem sa o realizam(daca se doreste realizarea diviziunii sau integrarii se introduce doar in caseta primului polinom). In caseta de Result se va afisa rezultatul operatiei. Cand dorim sa oprim aplicatia apasam pe butonul de x.

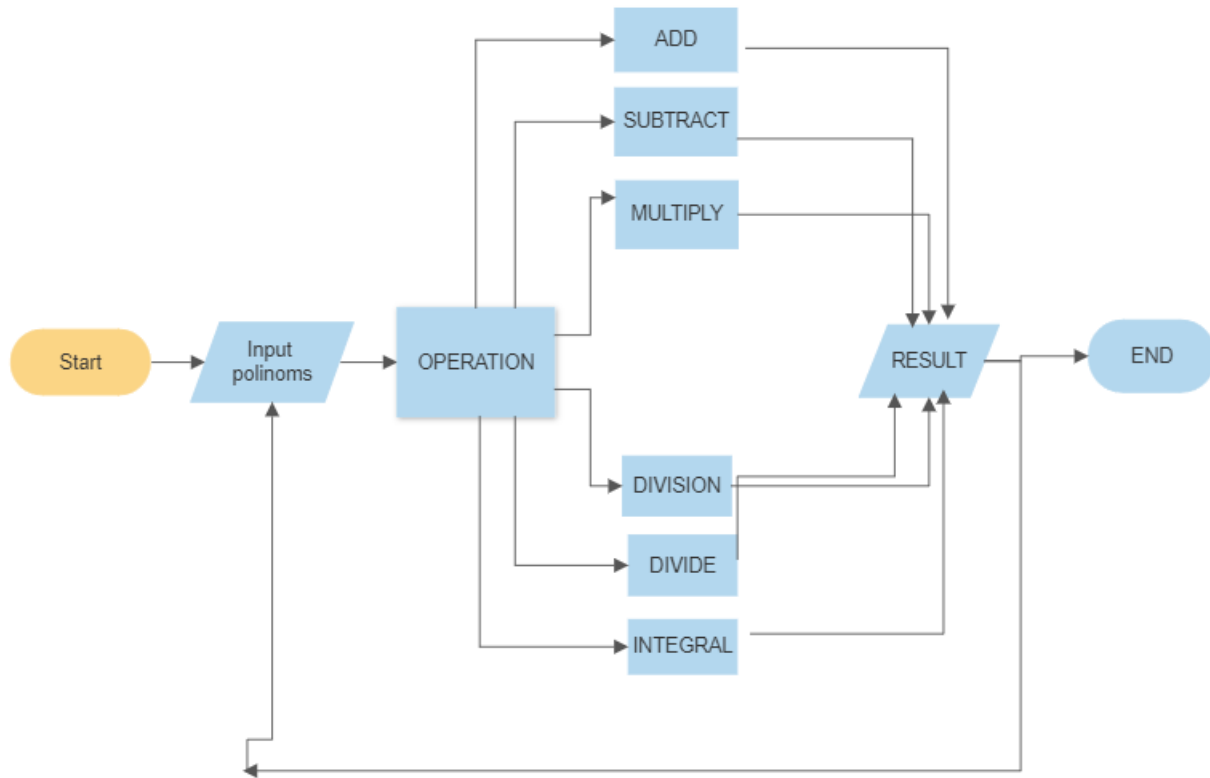
Daca dorim sa facem adunarea polinoamelor: $3x^2+4x^1$ si $2x^4+7x^2+9x^1$ vom urma pasii:

1. In prima caseta alaturata primului polinom introducem: $3x^2+4x^1$
2. In caseta 2 alaturata polinomului 2 introducem: $2x^4+7x^2+9x^1$
3. Apasam pe butonul ADD care va face adunarea
4. In dreapta lui Result: se va afisa rezultatul adunarii si anume $2.0x^4+10.0x^2+13.0x$

In acelasi mod se vor face si celelalte operatii, insa in cazul derivarii si integrarii, se va lua in considerare doar primul polinom.

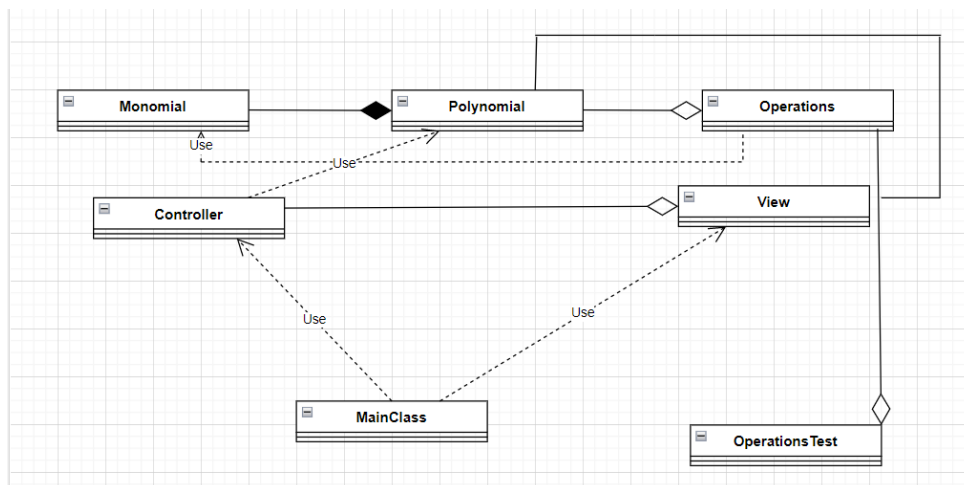
Use Case Diagram:





3. Proiectare

Diagrama UML de clase:



Diagramele pentru fiecare clasa in parte:

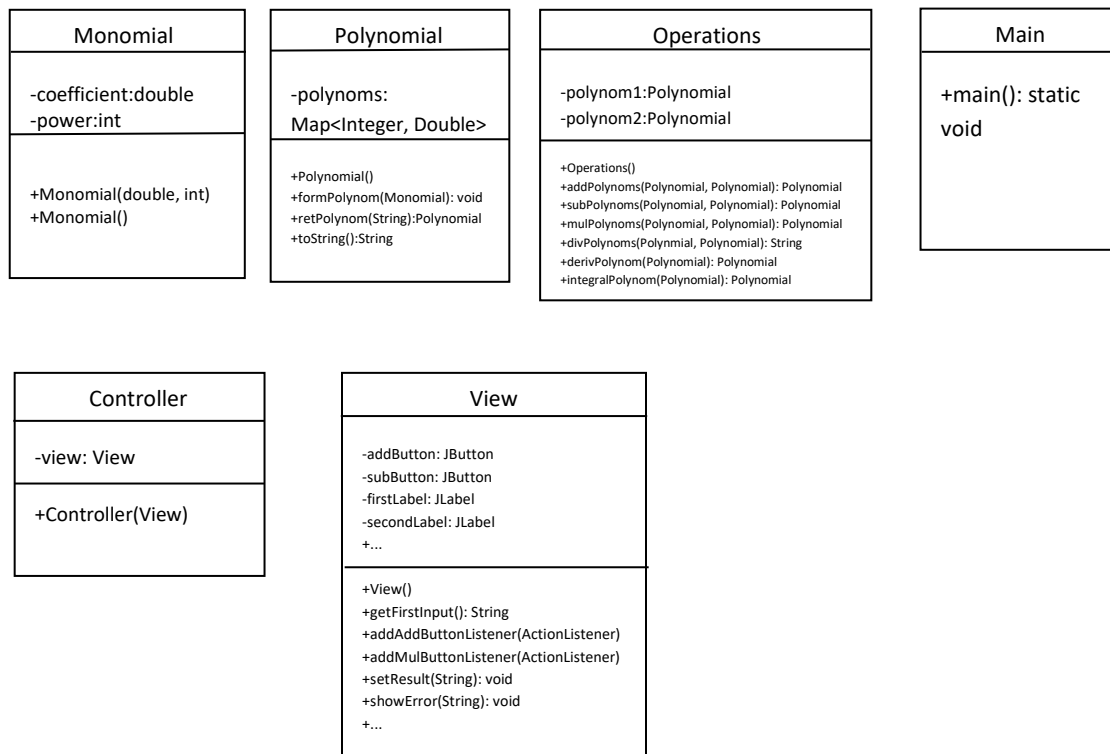
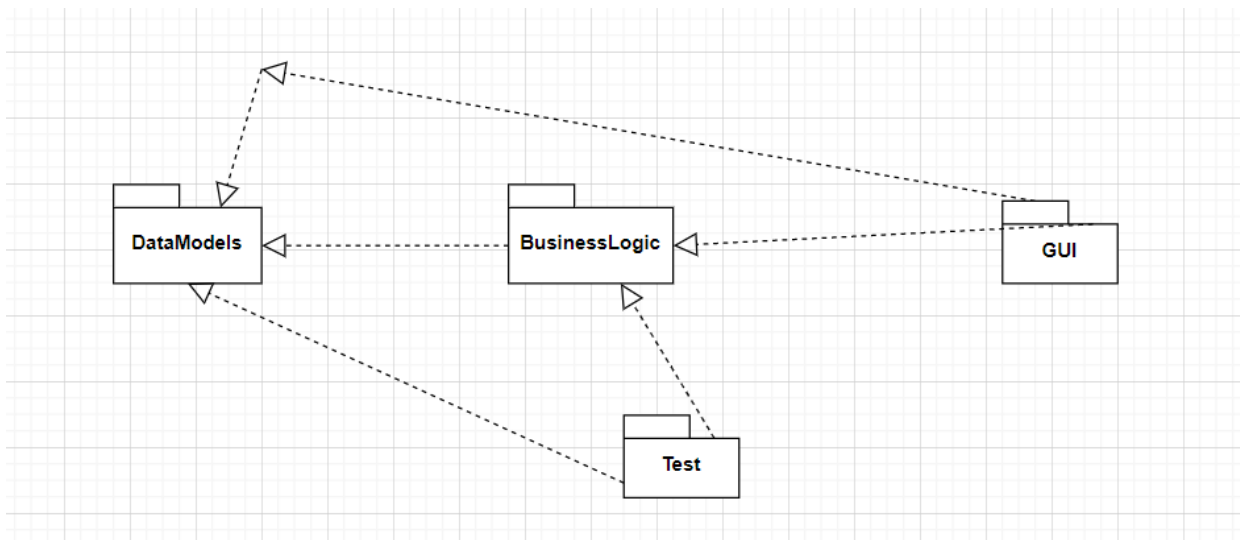


Diagrama de pachete:



Pentru realizarea operatiei de impartire din clasa Operation vom folosi pseudocodul din laborator:

Division of two polynomials

To divide two polynomials P and Q , the following steps should be performed:

Step 1 - Order the monomials of the two polynomials P and Q in descending order according to their degree.

Step 2 - Divide the polynomial with the highest degree to the other polynomial having a lower degree (let's consider that P has the highest degree)

Step 3 - Divide the first monomial of P to the first monomial of Q and obtain the first term of the quotient

Step 4 - Multiply the quotient with Q and subtract the result of the multiplication from P obtaining the remainder of the division

Step 5 - Repeat the procedure from step 2 considering the remainder as the new dividend of the division, until the degree of the remainder is lower than Q .

4. Implementare

Tema este structurata in pachete: DataModels, BusinessLogic, GUI, Test.

Pachetul DataModels contine clasele Monomial si Polynomial.

Clasa Monomial are ca si attribute puterea monomului(int power) si coeficientul acestuia(double coefficient). Atributele vor avea ca modificador de acces private pentru incapsularea datelor. Am creat doi constructori, unul cu parametrii, iar altul fara. Ca si metode avem getteri si setteri, setter pentru a fixa o valoare atributului, iar getter pentru a obtine valoarea acestuia intrucat modificadorul de acces este private.

Clasa Polynomial contine un HashMap de monoame, puterea reprezentand cheia, iar coeficientul valoarea. Aici se adauga pe rand monoamele ce formeaza polinomul cu ajutorul metodei formPolynom. Am creat un constructor fara parametrii in care se initializeaza HashMap-ul. Metodele pe care le am implementat aici sunt retPolynom care se foloseste de regex pentru a alege modelul polinoamelor(pattern) si pentru a imparti polinomul primit ca un string in grupuri astfel ca se obtine din primul grup coeficientul, iar din al doilea puterea. Astfel se formeaza un monom, iar cu ajutorul metodei formPolynom vom crea polinomul din monoame. Daca coeficientul si puterea nu sunt de tipul Double, respective Integer se va genera o exceptie si va aparea un mesaj de eroare pe ecran. Ultima metoda este toString() care afiseaza polinomul de la coeficientul cel mai mare la cel mai mic, luand in considerare si cazurile cand puterea e zero sau coeficientul e zero.

In pachetul BusinessLogic am creat clasa Operations. Aici vom regasi adunarea, scaderea, inmultirea, impartirea, derivarea si integrarea. Ca si attribute, am folosit doua polinoame: polynom1, polynom2 de tipul Polynomial. Am creat constructorul cu parametrii polynom1 si polynom2, apoi am realizat operatiile. Adunarea addPolynoms primeste doi parametrii de tipul Polynomial. Pentru inceput vom defini rezultatul de tip Polynomial care ii atribuim valoarea polynomului1 pe care il vom aduna cu celalalt. Se defineste si o variabila boolean pentru a testa daca puterile sunt egale sau nu. Cu ajutorul for-ului care parcurge map-ul polinomului 2 vom cauta puteri egale din primul si al doilea polinom, iar daca sunt egale setam booleanul pe true si cream un monom nou cu coeficientul

format din adunarea coeficientilor celor 2 monoame din for si a puterii egale in ambele parti. Acest mono mil adaugam in polinomul result pe care il vom returna la finalul metodei. Vom lua in considerare si cazul in care boolean-ul este false, deci nu se gaseste in polinomul 2 un termen cu puterea egala cu monomul current din primul polinom si il vom adauga la rezultat.

```
public Polynomial addPolynoms(Polynomial polynom1, Polynomial polynom2) {
    Polynomial result = new Polynomial();
    result = polynom1;
    boolean f;
    for (Map.Entry<Integer, Double> entry :
polynom2.getPolynoms().entrySet()) {
        f = false;
        int power = entry.getKey();
        double coeff = entry.getValue();
        for (Map.Entry<Integer, Double> entry1 :
polynom1.getPolynoms().entrySet()) {
            int powerOther = entry1.getKey();
            double coeffOther = entry1.getValue();

            if (power == powerOther) {
                f = true;
                Monomial m = new Monomial(coeff + coeffOther, power);
                result.formPolynom(m);
                break;
            }
        }
        if (f == false) {
            Monomial m = new Monomial(coeff, power);
            result.formPolynom(m);
        }
    }
    return result;
}
```

A doua operatie este scaderea-subPolynoms care primeste ca parametrii tot 2 polinoame si realizeaza scaderea acestora. Se parcurge polinomul 1, iar daca in polinomul 2 sunt termeni cu aceeasi putere se scad, altfel se pun la rezultat. Apoi vom parcurge al doilea polinom si vom adauga restul termenilor la rezultat, insa cu semnul minus.


```

public Polynomial subPolynoms(Polynomial polynom1, Polynomial polynom2) {
    Polynomial result = new Polynomial();

    for (Map.Entry<Integer, Double> entry :
polynom1.getPolynoms().entrySet()) {
        int power = entry.getKey();
        double coeff = entry.getValue();
        if (polynom2.getPolynoms().containsKey(power)) {
            double coeffOther = polynom2.getPolynoms().get(power);
            double newCoeff = coeff - coeffOther;
            if (newCoeff != 0) {
                Monomial m = new Monomial(newCoeff, power);
                result.formPolynom(m);
            }
        } else {
            Monomial m = new Monomial(coeff, power);
            result.formPolynom(m);
        }
    }
    for (Map.Entry<Integer, Double> entry :
polynom2.getPolynoms().entrySet()) {
        int power = entry.getKey();
        double coeff = entry.getValue();
        if (!polynom1.getPolynoms().containsKey(power)) {
            Monomial m = new Monomial(-coeff, power);
            result.formPolynom(m);
        }
    }
    return result;
}

```

Inmultirea polinoamelor se bazeaza pe parcurgerea polinoamelor cu un for in for care va inmulti primul monom al polinomului 1 cu toate monoamele din polinomul 2 si formarea unui rezultat intermediar pe care il vom adauga la rezultatul final pentru simplificarea formei finale astfel incat sa nu apara monoame cu aceeasi putere. Se va repeata procedeul pentru toate monoamele din polinomul 1.

```

public Polynomial mulPolynoms(Polynomial polynom1, Polynomial polynom2) {
    Polynomial result = new Polynomial();

    for (Map.Entry<Integer, Double> entry :
polynom1.getPolynoms().entrySet()) {
        Polynomial resultAux = new Polynomial();
        int power = entry.getKey();
        double coeff = entry.getValue();
        for (Map.Entry<Integer, Double> entry1 :
polynom2.getPolynoms().entrySet()) {
            int powerOther = entry1.getKey();
            double coeffOther = entry1.getValue();

            Monomial m = new Monomial(coeff * coeffOther, power +
powerOther);
            resultAux.formPolynom(m);
        }
        result = addPolynoms(resultAux, result);
    }
    return result;
}

```

Impartirea polinoamelor se bazeaza pe pseudocodul descris in laborator:

Division of two polynomials

To divide two polynomials P and Q , the following steps should be performed:

Step 1 - Order the monomials of the two polynomials P and Q in descending order according to their degree.

Step 2 - Divide the polynomial with the highest degree to the other polynomial having a lower degree (let's consider that P has the highest degree)

Step 3 - Divide the first monomial of P to the first monomial of Q and obtain the first term of the quotient

Step 4 - Multiply the quotient with Q and subtract the result of the multiplication from P obtaining the remainder of the division

Step 5 - Repeat the procedure from step 2 considering the remainder as the new dividend of the division, until the degree of the remainder is lower than Q .

```

public String divPolynoms(Polynomial polynom1, Polynomial polynom2) {
    String res = "";
    Polynomial quotient = new Polynomial();
    Polynomial remainder = new Polynomial();

    TreeMap<Integer, Double> reverseMap1 = new
TreeMap<>(Collections.reverseOrder());
    reverseMap1.putAll(polynom1.getPolynoms());
    TreeMap<Integer, Double> reverseMap2 = new
TreeMap<>(Collections.reverseOrder());
    reverseMap2.putAll(polynom2.getPolynoms());

    remainder.setPolynoms(reverseMap1);
    polynom2.setPolynoms(reverseMap2);

    while (Collections.max(remainder.getPolynoms().keySet()) >=
Collections.max(polynom2.getPolynoms().keySet())) {
        int powerP = Collections.max(remainder.getPolynoms().keySet());
        int powerQ = Collections.max(polynom2.getPolynoms().keySet());

        Polynomial aux = new Polynomial();
        double coeffP = reverseMap1.get(powerP);
        double coeffQ = reverseMap2.get(powerQ);

        if (coeffQ == 0 && powerQ == 0) {
            res = res + "You can't divide by 0!!!";
            return res;
        }

        Monomial m = new Monomial(coeffP / coeffQ, powerP - powerQ);
        aux.formPolynom(m);

        quotient.formPolynom(m);
        remainder = subPolynoms(remainder, mulPolynoms(aux, polynom2));
        if (remainder.toString() == "") {
            break;
        }
    }

    res = res + "Quotient: " + quotient.toString() + "      Remainder: " +
remainder.toString();
    return res;
}

```

Derivarea polinoamelor se face foarte usor. Aici am considerat polinomul pentru a realiza aceasta operatie. Rezultatul derivarii unui monom este coefficient * power * x^{power-1}. Cazul de exceptie este atunci cand puterea este 0, iar rezultatul derivarii este 1.

```

public Polynomial derivPolynom(Polynomial polynom1) {
    Polynomial result = new Polynomial();

    for (Map.Entry<Integer, Double> entry :
polynom1.getPolynoms().entrySet()) {
        int power = entry.getKey();
        double coeff = entry.getValue();
        if (power != 0) {
            Monomial m = new Monomial(coeff * (double) power, power - 1);
            result.formPolynom(m);
        }
    }
    return result;
}

```

Integrarea este inversul derivarii. Se aplica pe polinomul 1, iar rezultatul integrării unui monom este coefficient / (power+1) *x^power+1. Se adauga intr-un polinom toate rezultatele monoamelor integrate.

```

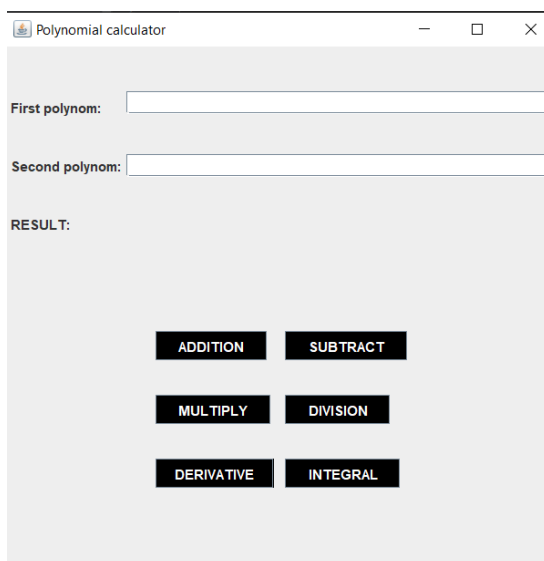
public Polynomial integralPolynom(Polynomial polynom1) {
    Polynomial result = new Polynomial();

    for (Map.Entry<Integer, Double> entry :
polynom1.getPolynoms().entrySet()) {
        int power = entry.getKey();
        double coeff = entry.getValue();

        Monomial m = new Monomial(coeff / (double) (power + 1), power + 1);
        result.formPolynom(m);
    }
    return result;
}

```

In pachetul GUI am realizat interfata grafica in clasa View.



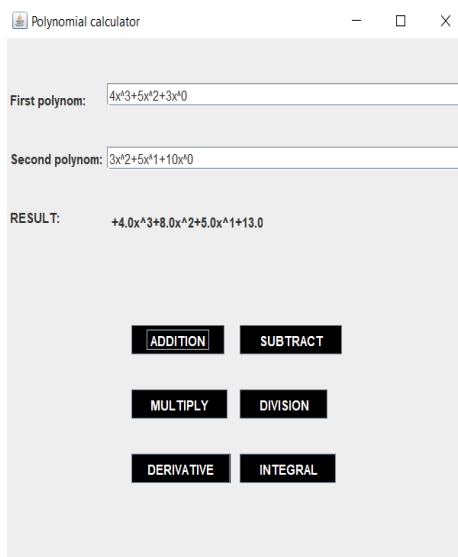
Am adaugat butoane pentru fiecare operatie in parte, am folosit 3 JLabel-uri pentru a afisa “First polynom”, “Second polynom”, “RESULT:” pentru a marca locul unde se vor introduce polinoamele si se va afisa rezultatul. Desigur ca acestea se vor introduce in niste JTextField-uri sub forma “ $ax^n+bx^{n-1}+cx^{n-2}+\dots+nx^0$ ”. In structurarea interfetei, am folosit panel-uri; in primul panel am adaugat label-urile de la first polynoms, second polynom si result, ordonandu-le pe axa y, apoi am facut alt panel pentru ordonarea pe axa y a casetelor de text si a label-ului de rezultat, urmand ca acestea sa fie introduse intr-un al treilea panel a partii de sus. La fel se procedeaza si pentru butoane, eu am ales sa adaug 3 butoane in stanga si 3 in partea dreapta, urmand ca apoi sa le introduc in alt panel al partii de jos. Panel-ul final este realizat pe axa y din panelul partii de sus si cel al partii de jos. Pe langa acestea, in clasa View am implementat metodele necesare pentru a asigura o functionalitate corecta. Metodele getFirstInput si getSecondInput vor extrage ceea ce vom introduce in dreapta lui First Polynom si Second Polynom si le va returna ca si string. Dupa aceea, avem metodele ActionListener pentru fiecare buton. Rezultatul operatiei se va incarca intr-un JLabel initial gol cu ajutorul metodei setResult. In clasa Main vom crea un Controller si un View si vom seta interfata sa fie vizibila pentru a o putea vedea.

In clasa Controller vom realiza functionalitatea butoanelor astfel incat pentru fiecare operatie se vor extrage polinoamele din View, cu ajutorul metodei retPolynom din clasa Polynomial implementata cu ajutorul regex-ului, apoi se va realiza operatia pentru fiecare buton si se va seta rezultatul.

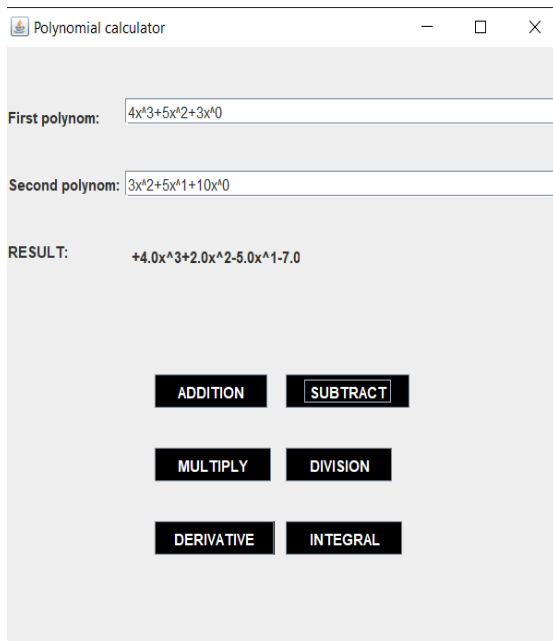
Ultima clasa implementata este TestOperations din pachetul test care va verifica pentru fiecare operatie in parte daca s-a realizat cu succes cu ajutorul lui AssertEquals sau AssertNotEquals. Se numara testele totale executate si numarul de teste executate cu success. @BeforeAll si @AfterAll se vor executa o singura data, iar @BeforeEach si @AfterEach inaintea si dupa fiecare test. Testele urmeaza dupa @Test si se executa pentru fiecare operatie. Acestea ne ajuta sa verificam functionalitatea corecta.

5. Rezultate

Pentru inceput sa verificam adunarea a doua numere. Sa luam de exemplu $4x^3+5x^2+3x^0 + 3x^2+5x^1+10x^0$ si sa verificam daca rezultatul adunarii este $+4.0x^3+8.0x^2+5.0x^1+13.0$.



Pentru scadere, vom demonstra ca functioneaza cum trebuie luand ca si exemplu urmatoarele polinoame: $4x^3+5x^2+3x^0$, $3x^2+5x^1+10x^0$, rezultatul asteptat fiind: $+4.0x^3+2.0x^2-5.0x^1-7.0$. Sa verificam acest lucru si in interfata:



Polynomial calculator

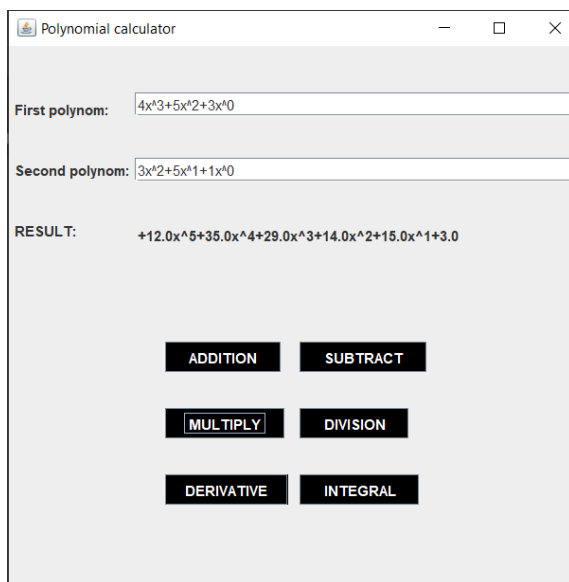
First polynomial: $4x^3+5x^2+3x^0$

Second polynomial: $3x^2+5x^1+10x^0$

RESULT: $+4.0x^3+2.0x^2-5.0x^1-7.0$

ADDITION SUBTRACT
 MULTIPLY DIVISION
 DERIVATIVE INTEGRAL

Inmultirea polinoamelor se poate verifica din exemplul urmator:



Polynomial calculator

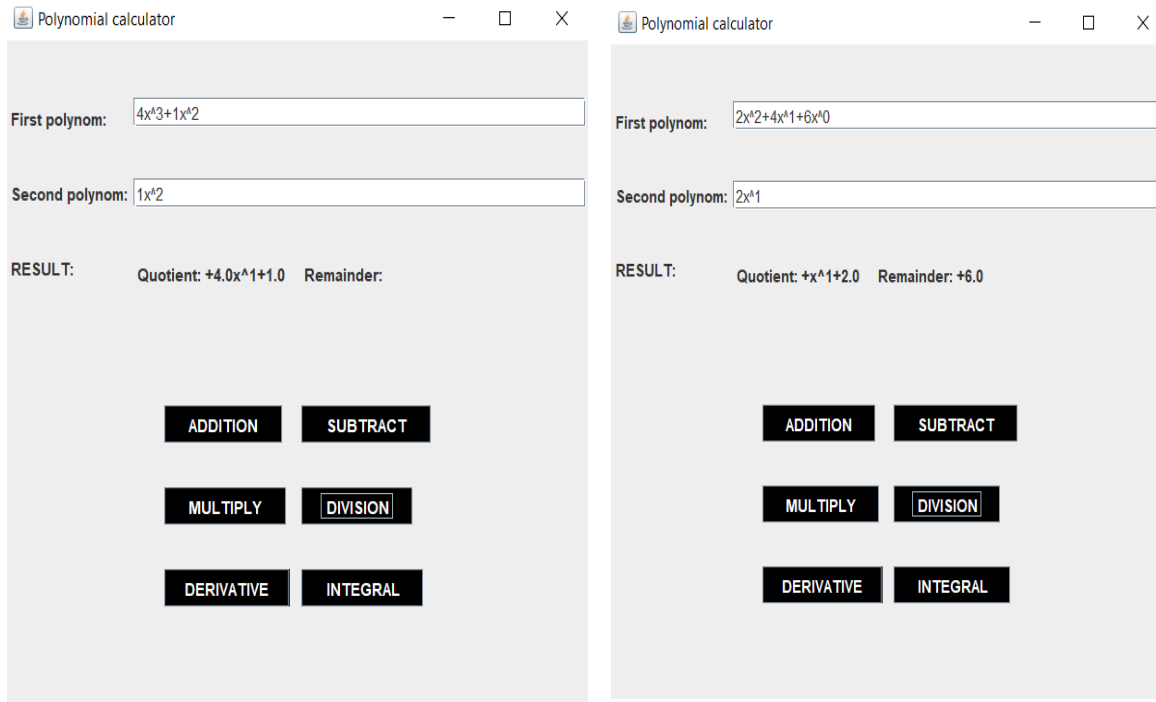
First polynomial: $4x^3+5x^2+3x^0$

Second polynomial: $3x^2+5x^1+1x^0$

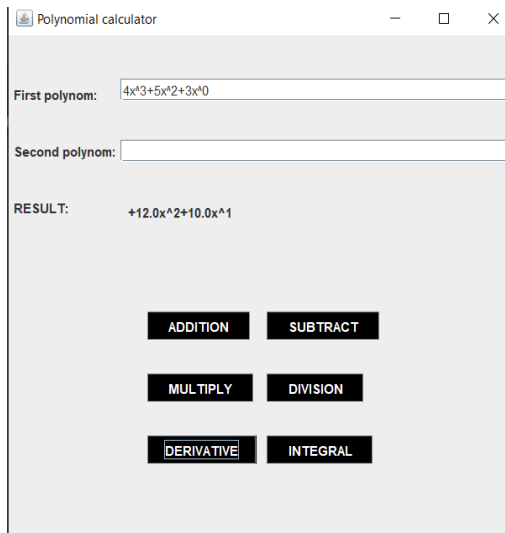
RESULT: $+12.0x^5+35.0x^4+29.0x^3+14.0x^2+15.0x^1+3.0$

ADDITION SUBTRACT
 MULTIPLY DIVISION
 DERIVATIVE INTEGRAL

Impartirea facuta dupa pseudocodul dat va afisa atat catul, cat si restul impartirii a doua polinoame. Se poate observa in exemplul urmator:



Derivarea polinoamelor se face foarte usor insa avem un caz cand nu respecta regula, puterea fiind zero.



Integrarea o voi reprezenta printr-un exemplu:

Polynomial calculator

First polynomial:

Second polynomial:

RESULT: **$+x^4+1.6666666666666667x^3+3.0x^1$**

ADDITION **SUBTRACT**

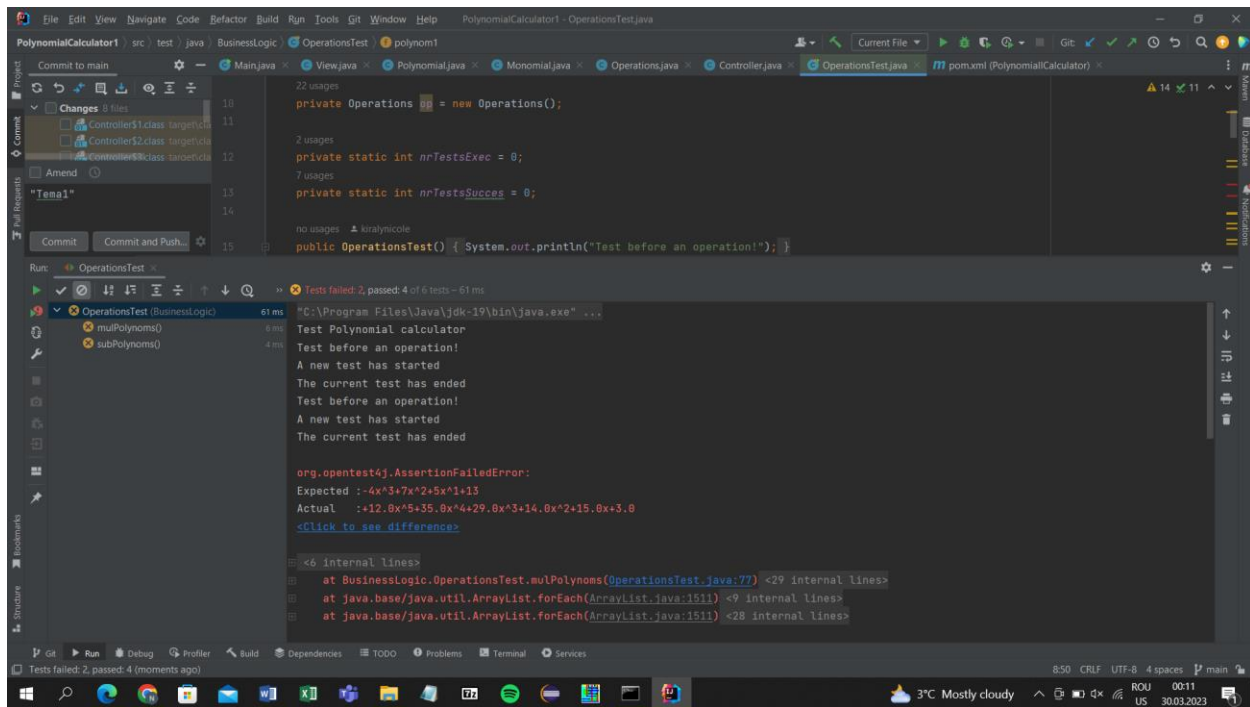
MULTIPLY **DIVISION**

DERIVATIVE **INTEGRAL**

Clasa OperationsTest testeaza fiecare operatie implementata si verifica cate teste au trecut.

The screenshot shows an IDE with the following components:

- Project Explorer:** Shows the project structure with folders like `src`, `main`, `BusinessLogic`, `DataModels`, `Monomial`, `Polynomial`, `GUI`, `Controller`, and `Main`.
- Code Editor:** Displays the `OperationsTest.java` file. The code includes:
 - Imports for `Polynomial` and `Monomial`.
 - A `test` method that creates a polynomial, performs a derivative operation, and asserts the result.
 - An `@Test` method named `integralPolynom` that performs an integration operation and asserts the result.
- Run Console:** Shows the output of the test execution. It indicates that 6 tests passed out of 6 tests, with a total time of 553 ms. The output also shows the state of the test runner, including the number of tests executed and successful.



6. Concluzii

În această temă, am avut posibilitatea să învăț cum se realizează o interfață grafică de la zero, având libertatea să aleg un design pentru aceasta. De asemenea, am învățat cum se pot face operații matematice pe polinoame în cod, găsind cazuri de excepție unde nu urma aceeași regulă. Mai mult decât atât, am recapitulat principiile OOP învățate în semestrul 1 și le-am fixat astfel încât acum e mult mai ușor de lucrat în JAVA. Am folosit debugger-ul de o multitudine de ori pentru a-mi verifica cazurile speciale. În final, testarea unitară a fost de asemenea un concept nou cu care nu am lucrat până acum.

Posibile dezvoltări ulterioare ale calculatorului polinomial:

- adaugarea unui nou buton care face o operație cu el însuși
- aparitia graficului pe axele de coordonate la apăsarea unui buton
- găsirea rădăcinilor polinomului

7. Bibliografie

- https://www.smartdraw.com/flowchart/flowchart-maker.htm?id=45057&gclid=Cj0KCOjwT_qgBhDFARIsABcDjOeDxPnrmRKpzk0c39T5ye_RqW4KpiAqJ_Ei-F9FIW2hSFYK9WV5LMsaAi8GEALw_wcB
- <https://dsrl.eu/courses/pt/materials/lectures/> *Lecture1, Lecture 2*
- <https://regexr.com/>
- https://www.w3schools.com/java/java_regex.asp
- <https://users.utcluj.ro/~igiosan/Resources/POO/Curs/POO09.pdf>