DOCUMENTATIE

TEMA 1

NUME STUDENT: Vasilachi Eugen
GRUPA: 30227

CUPRINS

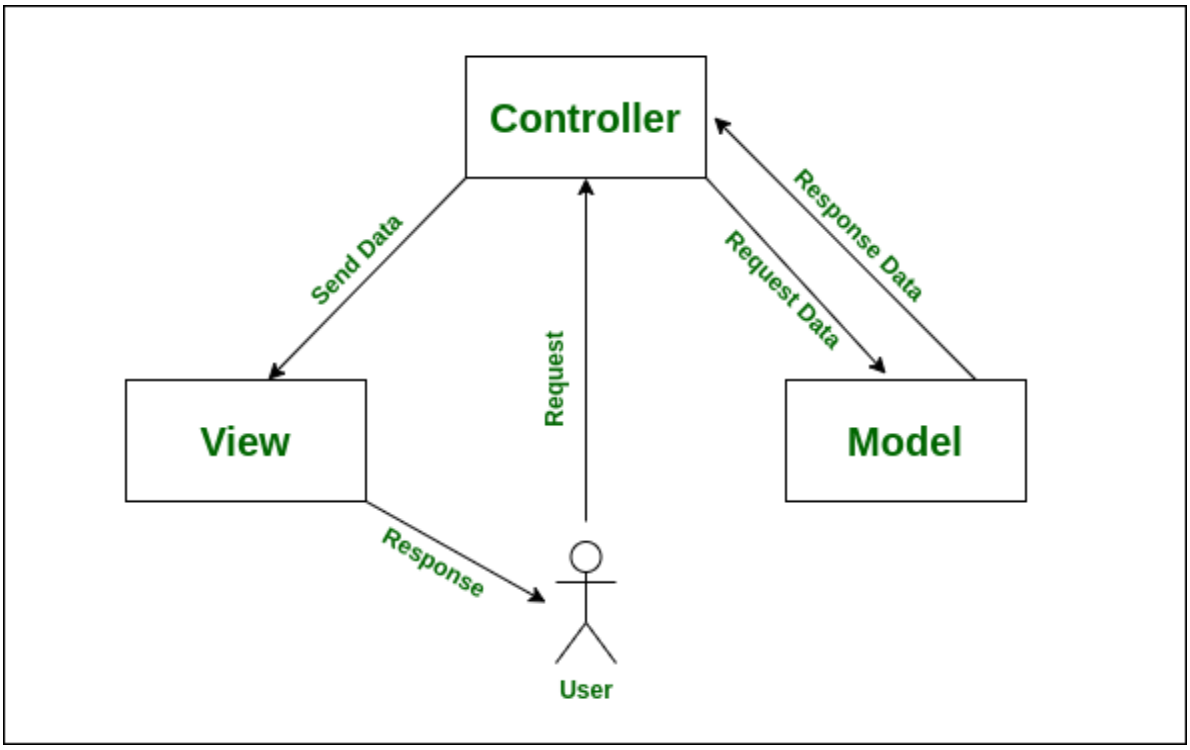
1. Obiectivul temei

Obiectivul principal temei reprezinta realizarea unui calculator care poate sa efectueze calcule pe polinoame, cum ar fi: adunarea, scaderea, inmultirea, impartirea a doua polinoame, derivata si integrala unui polinom.

Obiectivele secundare ce trebuie urmate pentru indeplinirea obiectivului principal sunt:

- Crearea unui GUI (graphical user interface) pentru a face posibila interactionarea cu user-ul;
- Realizarea unui model arhitectural MVC (Model View Controller) si a pachetelor pentru fiecare parte din model;
- Conectarea Modelului cu View-ul prin intermediul Controller-ului.

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



Analiza MVC-ului:

User-ul va introduce polinomul/polinoamele pe care vrea sa efectueze operatiile. Controller-ul are rolul de a lua acele informatii si de a le transmite la Model care va efectua operatiile. Controller-ul va primi mai apoi rezultatele de la Model pe care le va transmite View-ului care va raspunde cererii user-ului.

MVC-ul reprezinta un bun model arhitectural deoarece te ajuta sa iti grupezi clasele si pachetele dupa functionalitatea lor.

Cerinte functionale:

1. Utilizarea unui design OOP (incapsulare, decompozitie, mostenire, polimorfism)

2. Utilizarea LinkedList in loc de ArrayList

3. Utilizarea foreach in loc de for cu indecsi

4. Implementarea unei Interfete Grafice cu Utilizatorul folosind Java Swing sau JavaFX

5. Implementarea operatiilor de adunare, scadere, inmultire si impartire a doua polinoame

6. Implementarea operatiilor de derivare si integrare a unui polinom.

7. Folosirea expresiilor regulate si potrivirea modelului pentru extragerea coeficientilor polinomului.

8. Utilizarea framework-ului JUnit pentru testare

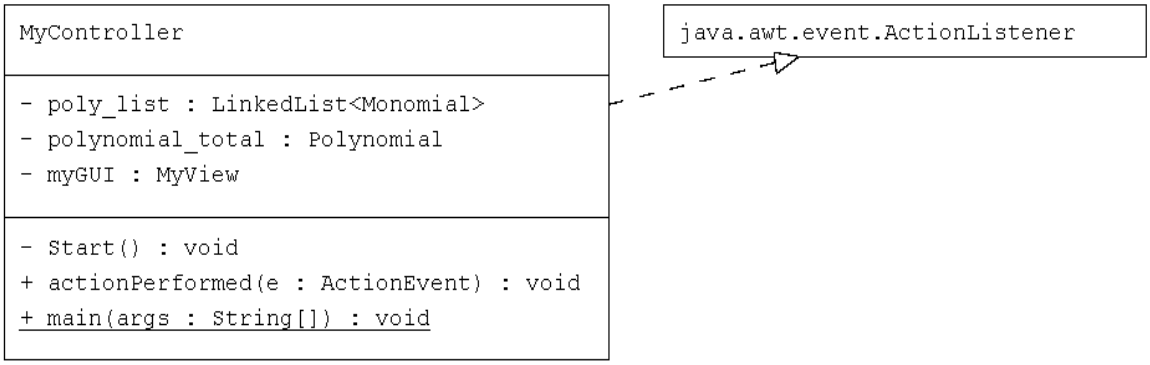
3.Proiectare

Prezentarea diagramelor UML pentru fiecare pachet:

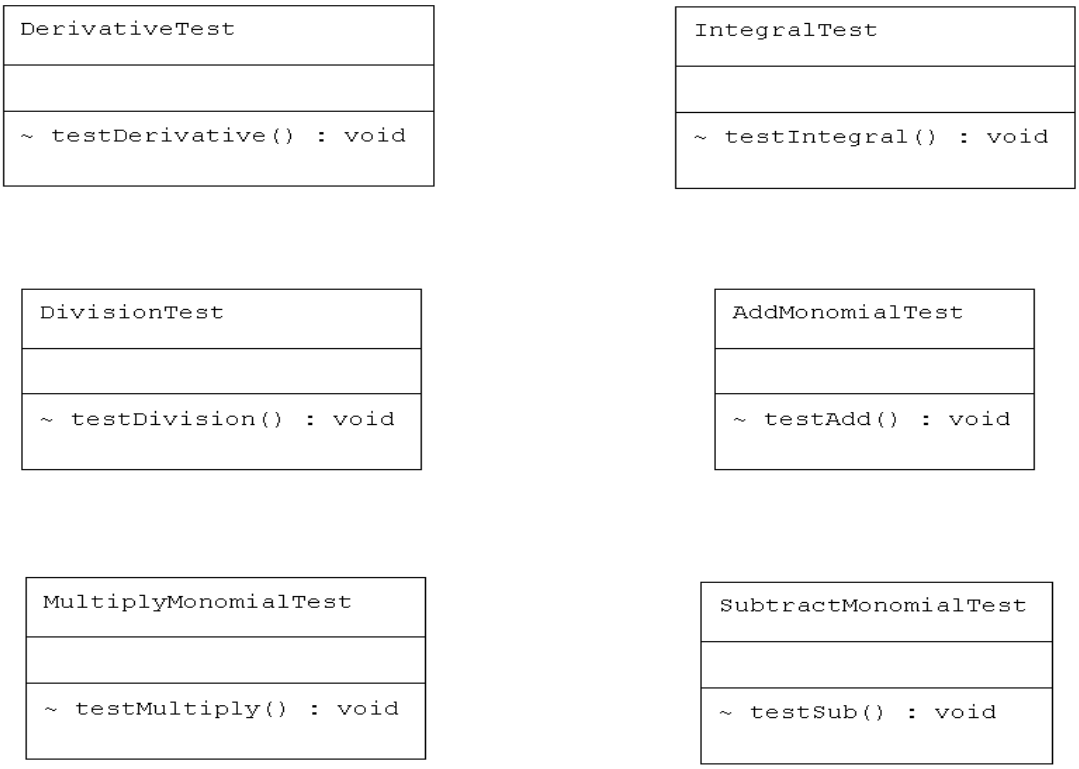
Pachetul model:

Pachetul view:

Pachetul controller:



Pachetul test:



Fiecare polinom este alcatuit dintr-o serie de monoame. Clasa Monomial defineste atributele si metodele specifice unui monom: un coeficient real, un exponent intreg, metode de get si set pentru incapsularea atributelor si o metoda toString() care va afisa monomul in functie de coeficientul si exponentul sau.

Clasa Polynomial retine o lista (LinkedList) de monoame specifica fiecarui obiect de tip Polynomial. Metodele implementate in aceasta clasa sunt de fapt operatiile pe polinoame: adunare, scadere, inmultire, impartire, derivata si integrala. Pentru o mai buna structurare a codului, s-a creat cate o clasa pentru fiecare operatie in parte.

4. Implementare

1) Clasa Monomial

```
package model;

public class Monomial {
    private float coefficient;
    private int exponent;

    public Monomial(float coefficient, int exponent) {
        this.coefficient = coefficient;
        this.exponent = exponent;
    }
    public float getCoefficient(){
        return coefficient;
    }
    public int getExponent(){
        return exponent;
    }
    public void setCoefficient(float coefficient){
        this.coefficient = coefficient;
    }
    public void setExponent(int exponent){
        this.exponent = exponent;
    }
    public String toString(){
        String mono;
        switch(exponent){
            case 0:
                if (coefficient == (int) coefficient) {
                    mono = String.valueOf((int)coefficient);
                    return mono;
                }
```

```
        else {
            mono = String.format("%.2f", coefficient);   // float with 2 decimals
            return mono;
        }
    case 1:
        if(coefficient == 1.0) {
            mono = "x";
            return mono;
        }
        else if(coefficient == -1.0) {
            mono = "-x";
            return mono;
        }
        else {
            if (coefficient == (int) coefficient) {
                mono = String.valueOf((int)coefficient) + "x";
                return mono;
            }
            else {
                mono = String.valueOf(coefficient) + "x";
                return mono;
            }
        }
    default:
        if(coefficient == 1.0) {
            mono = "x^" + String.valueOf(exponent);
            return mono;
        }
        else if(coefficient == -1.0) {
            mono = "-x^" + String.valueOf(exponent);
            return mono;
```

```
            }
            else if(coefficient == -1.0) {
                mono = "-x^" + String.valueOf(exponent);
                return mono;
            }
            else {
                if (coefficient == (int) coefficient) {
                    mono = String.valueOf((int)coefficient) + "x^" + String.valueOf(exponent);
                    return mono;
                }
                else {
                    mono = String.format("%.2f", coefficient) + "x^" + String.valueOf(exponent);  // float with 2 decimals
                    return mono;
                }
            }
    }
}
public Monomial multiply(Monomial m1, Monomial m2) {
    float coeff = m1.getCoefficient() * m2.getCoefficient();
    int exp = m1.getExponent() + m2.getExponent();
    return new Monomial(coeff, exp);
}
```

Metoda toString() creeaza un string format din coeficient, x si exponent. Daca coeficientul este -1 sau 1 atunci nu-l mai afiseaza, altfel va afisa un numar real in 2 zecimale. La fel se intampla si daca exponentul este 1, iar daca este 0 atunci se va afisa doar coeficientul.

2) Clasa Polynomial

```java
package model;

import java.util.LinkedList;

public class Polynomial {
    private LinkedList<Monomial> poly;

    public Polynomial(){
        poly = new LinkedList<>();
    }
    public Polynomial(Monomial mono) {
        poly = new LinkedList<>();
        poly.add(mono);
    }
    public Polynomial(LinkedList<Monomial> list) {
        poly = list;
    }

    public LinkedList<Monomial> getPoly(){
        return poly;
    }
    public void setPoly(LinkedList<Monomial> poly){
        this.poly = poly;
    }


    public void handleInput(String input) {
        if(input.equals(""))
            return;
        String[] poly_str = input.split( regex: "(?=[+-])");
        create_list(poly_str);
```

```
    public void create_list(String[] poly_str) {
        for(String str : poly_str) {
            String[] split_str = str.split( regex: "[x^]");
            if(split_str.length == 3) {
                String s = split_str[0];
                switch(s){
                    case "-":
                        Monomial m1 = new Monomial( coefficient: -1, Integer.parseInt(split_str[2]));
                        poly.add(m1);
                        break;
                    case "+":
                        Monomial m2 = new Monomial( coefficient: 1, Integer.parseInt(split_str[2]));
                        poly.add(m2);
                        break;
                    case "":
                        Monomial m3 = new Monomial( coefficient: 1, Integer.parseInt(split_str[2]));
                        poly.add(m3);
                        break;
                    default:
                        Monomial m4 = new Monomial(Integer.parseInt(split_str[0]), Integer.parseInt(split_str[2]));
                        poly.add(m4);
                        break;
                }
            }
            else if(split_str.length == 1){
                if(str.contains("x")){
                    switch(split_str[0]){
                        case "-":
                            Monomial m1 = new Monomial( coefficient: -1,  exponent: 1);
                            poly.add(m1);
                            break;
```

```
            else if(split_str.length == 1){
                if(str.contains("x")){
                    switch(split_str[0]){
                        case "-":
                            Monomial m1 = new Monomial( coefficient: -1,  exponent: 1);
                            poly.add(m1);
                            break;
                        case "+":
                            Monomial m2 = new Monomial( coefficient: 1,  exponent: 1);
                            poly.add(m2);
                            break;
                        default:
                            Monomial m3 = new Monomial(Integer.parseInt(split_str[0]),  exponent: 1);
                            poly.add(m3);
                            break;
                    }
                }
                else {
                    Monomial mono = new Monomial(Integer.parseInt(split_str[0]),  exponent: 0);
                    poly.add(mono);
                }
            }
            else if(split_str.length == 0){      /// Polynomial which starts with 'x'
                Monomial mono = new Monomial( coefficient: 1,  exponent: 1);
                poly.add(mono);
            }
        }
    }
    public LinkedList<Monomial> add(Polynomial p) {
        for(Monomial m : p.getPoly()) {
            AddMonomial.addMonomial( p: this, m);
```

```java
    public LinkedList<Monomial> add(Polynomial p) {
        for(Monomial m : p.getPoly()) {
            AddMonomial.addMonomial( p: this, m);
        }
        return poly;
    }
    public LinkedList<Monomial> sub(Polynomial p) {
        for(Monomial m : p.getPoly()) {
            SubtractMonomial.subMonomial( p: this, m);
        }
        return poly;
    }
    public LinkedList<Monomial> multiply(Polynomial p) {
        //poly = MultiplyMonomial.multiply(this, p);
        //return poly;
        return MultiplyMonomial.multiply(this, p);
    }
    public void derivative() {
        LinkedList<Monomial> newPoly = new LinkedList<>();
        for(Monomial mono : poly) {
            if(mono.getExponent() > 1) {
                Monomial newMono = new Monomial( coefficient: mono.getCoefficient() * mono.getExponent(), exponent: mono.getExponent() - 1);
                newPoly.add(newMono);
            }
            else if(mono.getExponent() == 1) {
                Monomial newMono = new Monomial(mono.getCoefficient(), exponent: 0);
                newPoly.add(newMono);
            }
        }
        poly = newPoly;
    }
```

```java
        if(!p.getPoly().isEmpty())
            reminder.setPoly(p.getPoly());
        Polynomial[] result = new Polynomial[2];
        result[0] = quotient;
        result[1] = reminder;
        return result;
    }
    public String toString(){
        if(poly.isEmpty())
            return "";
        String str = "";
        for(Monomial m : poly) {
            if(m.getCoefficient() == 0)
                continue;
            if(m != poly.getFirst() && m.getCoefficient() > 0)
                str += "+" + m.toString();
            else str += m.toString();
        }
        if(str.equals(""))
            return "0";
        return str;
    }
}
```

Un obiect de tipul Polynomial va retine o lista de obiecte de tipul Monomial. In aceasta clasa sunt apelate fiecare dintre operatiile specifice pe polinoame. Pentru fiecare operatie s-a creat o clasa separata care implementeaza o metoda ce urmeaza sa fie apelata in clasa Polynomial.

Metoda handleInput(String input) creeaza un array de string-uri din input, avand ca delimitator + sau -, apoi apeleaza metoda create_list ce are ca parametru array-ul de string-uri. In functie de lungimea fiecarui string din acest array, metoda instantiaza obiecte Monomial.

Metoda add(Polynomial p) parcurge toate monoamele din p si apeleaza metoda statica addMonomial din clasa AddMonomial care pentru fiecare monom din p se va duce in lista de monoame din obiectul this si daca gaseste un element cu acelasi exponent, atunci actualizeaza coeficientul acelui element cu suma dintre cei doi coeficienti. Daca s-a ajuns la finalul listei si inca nu s-a efectuat operatia de adunare, atunci adauga elementul la sfarsitul listei. Altfel, daca exponentul elementului se afla intre 2 exponenti din lista, va insera elementul intre cele 2 monoame.

Metoda sub(Polynomial p) face cam acelasi lucru cu metoda add, insa daca a gasit exponenti egali, atunci va scadea coeficientii.

Metoda multiply(Polynomial p1, Polynomial p2) pentru fiecare combinatie de mono1, mono2, mono1 facand parte din lista lui p1 si mono2 din lista lui p2, apeleaza metoda multiply specifica fiecarui monom in care se inmultesc coeficientii si se aduna exponentii.

Metoda derivative() creeaza o noua lista, transforma coeficientii si exponentii monoamelor din lista obiectului this astfel: pentru fiecare monom care are exponentul mai mare decat 1 se creeaza un nou monom care sa aiba coeficientul produsul dintre vechiul coeficient si vechiul exponent si exponentul mai mic cu 1; daca exponentul este 1, atunci nou monom va avea acelasi coeficient si exponentul 0. Metoda face ca lista obiectului this sa pointeze la noua lista creata.

Metoda integral() este asemanatoare cu derivative(), insa cand exponentul este 0, atunci noul monom acelasi coeficient si exponentul egal cu 1. Atunci cand exponentul

este mai mare decat 0, noul coeficient va lua valoarea raportului dintre vechiul coeficient si vechiul exponent +1 si noul coeficient va creste cu o unitate.

Metoda division(Polynomial q) returneaza un array de 2 elemente format din catul si restul impartirii a 2 polinoame. Polinomul diffPoly este un polinom egal cu produsul dintre monomul current din cat si q si este un polinom intermediar de care ne vom ajuta ca sa aflam noul p ce va fi egal cu diferenta dintre vechiul p si diffPoly. Atunci cand se ajunge ca noul p sa aiba gradul mai mic decat cel al lui q, algoritmul se opreste, catul calculandu-se in bucla while si restul fiind p-ul ramas.

Metoda toString() apeleaza pentru fiecare monom din lista toString() din Monomial, facand legatura intre ele.

5. Rezultate

assertEquals primeste 2 parametri: valoarea asteptata si valoarea calculate. Va returna un succes daca cele 2 valori coincid.

```java
package test;

import model.Polynomial;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class AddMonomialTest {

    @Test
    void testAdd() {
        var p1 = new Polynomial();
        var p2 = new Polynomial();
        p1.handleInput("x+1");
        p2.handleInput("x+2");
        p1.add(p2);
        assertEquals( expected: "2x+3", String.valueOf(p1));
    }
}
```

```java
package test;

import model.Polynomial;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class SubtractMonomialTest {
    @Test
    void testSub() {
        var p1 = new Polynomial();
        var p2 = new Polynomial();
        p1.handleInput("2x+1");
        p2.handleInput("x+2");
        p1.sub(p2);
        assertEquals( expected: "x-1", String.valueOf(p1));
    }
}
```

```java
package test;

import model.Monomial;
import model.Polynomial;
import org.junit.jupiter.api.Test;

import java.util.LinkedList;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class MultiplyMonomialTest {
    @Test
    void testMultiply() {
        var p1 = new Polynomial();
        var p2 = new Polynomial();
        p1.handleInput("x+1");
        p2.handleInput("x+2");
        LinkedList<Monomial> result = p1.multiply(p2);
        Polynomial p3 = new Polynomial(result);
        assertEquals( expected: "x^2+3x+2", String.valueOf(p3));
    }
}
```

```java
package test;

import model.Polynomial;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class DivisionTest {
    @Test
    void testDivision() {
        var p1 = new Polynomial();
        var p2 = new Polynomial();
        p1.handleInput("x^2-1");
        p2.handleInput("x+1");
        Polynomial[] p3;
        p3 = p1.division(p2);
        assertEquals( expected: "x-1", String.valueOf(p3[0]));
        assertEquals( expected: "0", String.valueOf(p3[1]));
    }
}
```

```java
package test;

import model.Polynomial;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class DerivativeTest {
    @Test
    void testDerivative() {
        var p1 = new Polynomial();
        p1.handleInput("x^2+2x+1");
        p1.derivative();
        assertEquals( expected: "2x+2", String.valueOf(p1));
    }
}
```

```java
package test;

import model.Polynomial;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class IntegralTest {
    @Test
    void testIntegral() {
        var p1 = new Polynomial();
        p1.handleInput("2x^2+2x+1");
        p1.integral();
        assertEquals( expected: "0.67x^3+x^2+x", String.valueOf(p1));
    }
}
```

6. Concluzii

O imbunatatire a eficientei ar fi utilizarea cautarii binare atunci cand cautam un monom cu un anumit exponent in lista noastra de monoame.

Aceasta tema m-a ajutat sa invat cum sa fac operatii pe string-uri in Java (ex. split), sa folosesc modelul arhitectural MVC, sa folosesc Gitlab-ul si operatiile aferente (add, commit, push) si sa utilizez framework-ul JUnit pentru testare.

7. Bibliografie

https://docs.oracle.com/javase/tutorial/uiswing/

https://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm

https://www.baeldung.com/javafx

https://www.vogella.com/tutorials/JUnit/article.html

https://www.baeldung.com/junit-5

https://google.github.io/styleguide/javaguide.html