2019

SISTEM DE PROCESARE A POLINOAMELOR

Tehnici De Programare

Alexandru Farcas

1 CUPRINS

2 OBIECTIVUL TEMEI

Obiectivul principal al primei teme de laborator îl reprezintă propunerea, proiectarea si implementarea unui sistem de procesare al polinoamelor de u singura variabila cu coeficienți întregi.

Obiectivele secundare care trebuie atinse in vederea îndeplinirii obiectivului principal sunt următoarele:

- Implementarea operației de înmulțire
- Implementarea operației de împărțire
- Implementarea operației de derivare
- Implementarea operației de integrare
- Testarea aplicației realizate prin intermediul JUnit

Cerințe suplimentare:

- Trebuie respectate paradigmele POO:
  - clase Polinom, Monom
  - folosirea List<> în loc de array[]
  - folosirea foreach în loc de for(int i = 0…)
- Aplicația trebuie organizata pe pachete
- Trebuie respectate convențiile de nume Java
- Trebuie implementata o interfața grafica fără utilizarea instrumentelor de tip ,,Drag and Drop”
- Trebuie furnizata si o documentație corespunzătoare proiectului realizat
- Tema trebuie predata si pe contul de bitbucket utcn_dsrl conform indicațiilor din descrierea Laboratorului

3 ANALIZA PROBLEMEI, MODELARE, SCENARII, CAZURI DE UTILIZARE

După cum impune cerința, tema abordata are la baza expresii polinomiale de o singura variabilă, cu coeficienți întregi.

În matematică, un polinom este o expresie construită dintr-una sau mai multe variabile și constante, folosind doar operații de adunare, scădere, înmulțire și ridicare la putere constantă pozitivă întreagă.

Polinoamele sunt construite din termeni numiți monoame, care sunt alcătuite dintr-o constantă (numită coeficient) înmulțită cu una sau mai multe variabile. Fiecare variabilă poate avea un exponent constant întreg pozitiv. Exponentul unei variabile dintr-un monom este egal cu gradul acelei variabile în acel monom. Un monom fără variabile se numește monom constant, sau doar constantă. Gradul unui termen constant este 0. Coeficientul unui monom poate fi orice număr, inclusiv fracții, numere iraționale sau negative, însă tema prezentata are la baza doar coeficienți întregi, chiar daca efectuarea anumitor operații asupra acestora poate conduce la coeficienți reali.

Un polinom construit cu o singură variabilă se numește univariat.

Toate polinoamele de o variabilă sunt echivalente cu un polinom de forma:

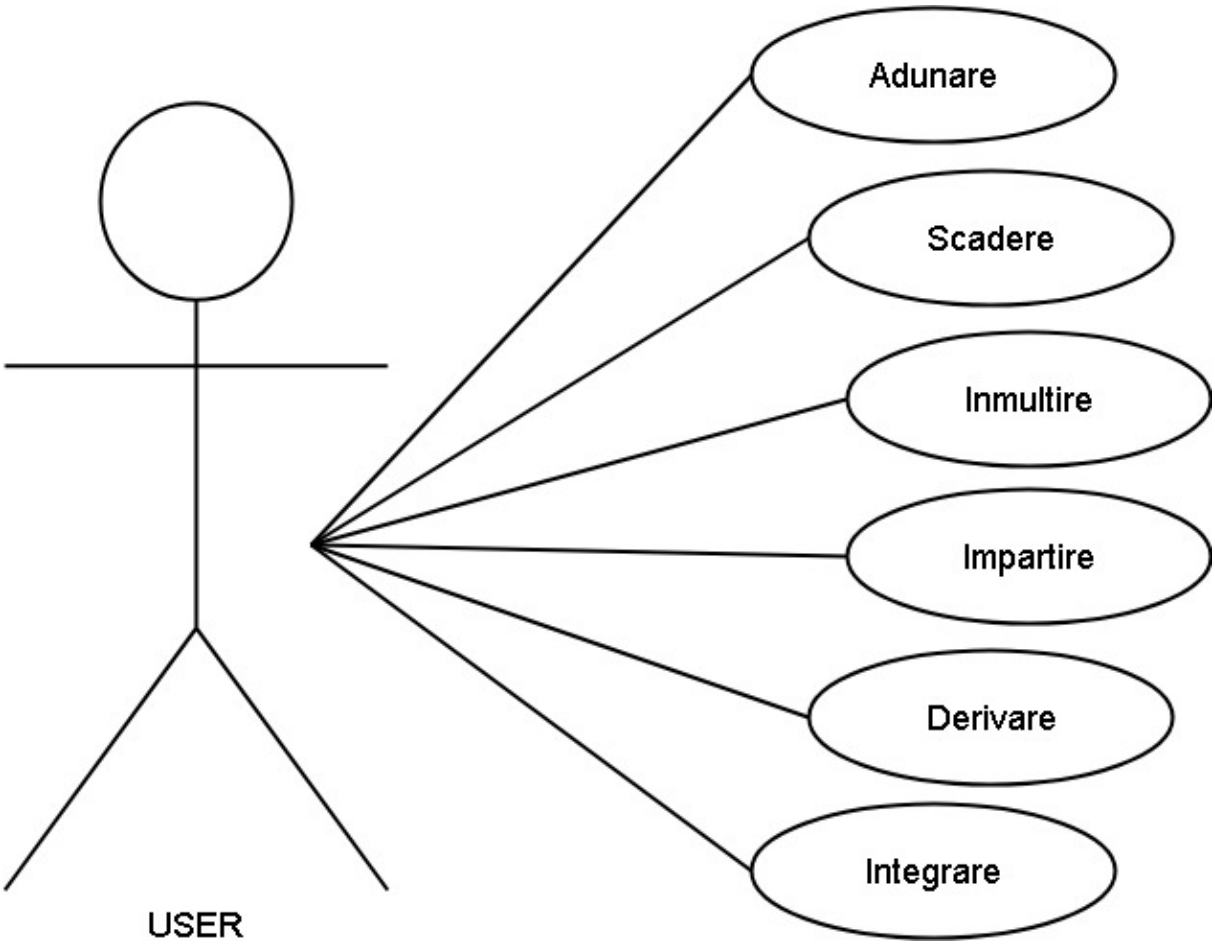
$$a^n x^n + a^{n-1} x^{n-1} + \cdots + a^2 x^2 + a^1 x^1 + a^0$$

Această forma este considerată forma generală a polinoamelor de o singură variabilă.

Asupra polinoamelor pot fi efectuate toate operațiile fundamentale, respectiv:

- Adunare
- Scădere
- Înmulțire
- Împărțire
- Derivare
- Integrare

Astfel, utilizatorului i se pun la dispoziție toate aceste operații fundamentale, prezentate in următoarea diagrama *„Use Case”:*

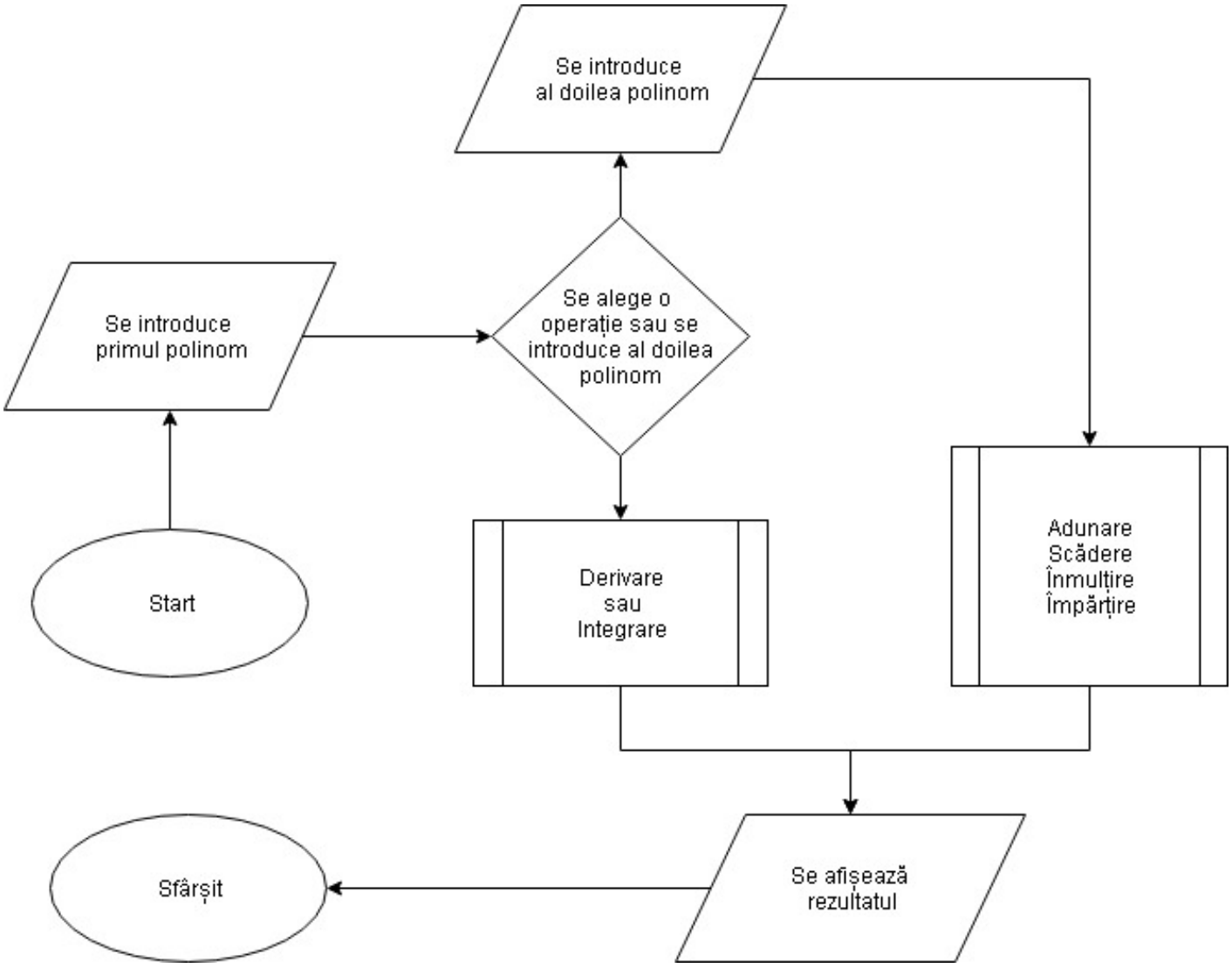


Pentru a efectua una dintre operațiile mai sus menționate, utilizatorul trebuie sa urmeze următorul proces:

- Inițial, utilizatorul introduce prima expresie polinomiala de la tastatura
- Pentru a efectua operațiile de derivare sau de integrare, un singur polinom este de ajuns, astfel, utilizatorul poate obține rezultatul prin apăsarea butonului respectiv, imediat după introducerea primului polinom.
- Celelalte operații necesita doi operanzi, astfel, utilizatorul este nevoit sa introducă un nou polinom pentru ca ulterior sa fie calculat rezultatul operației alese.

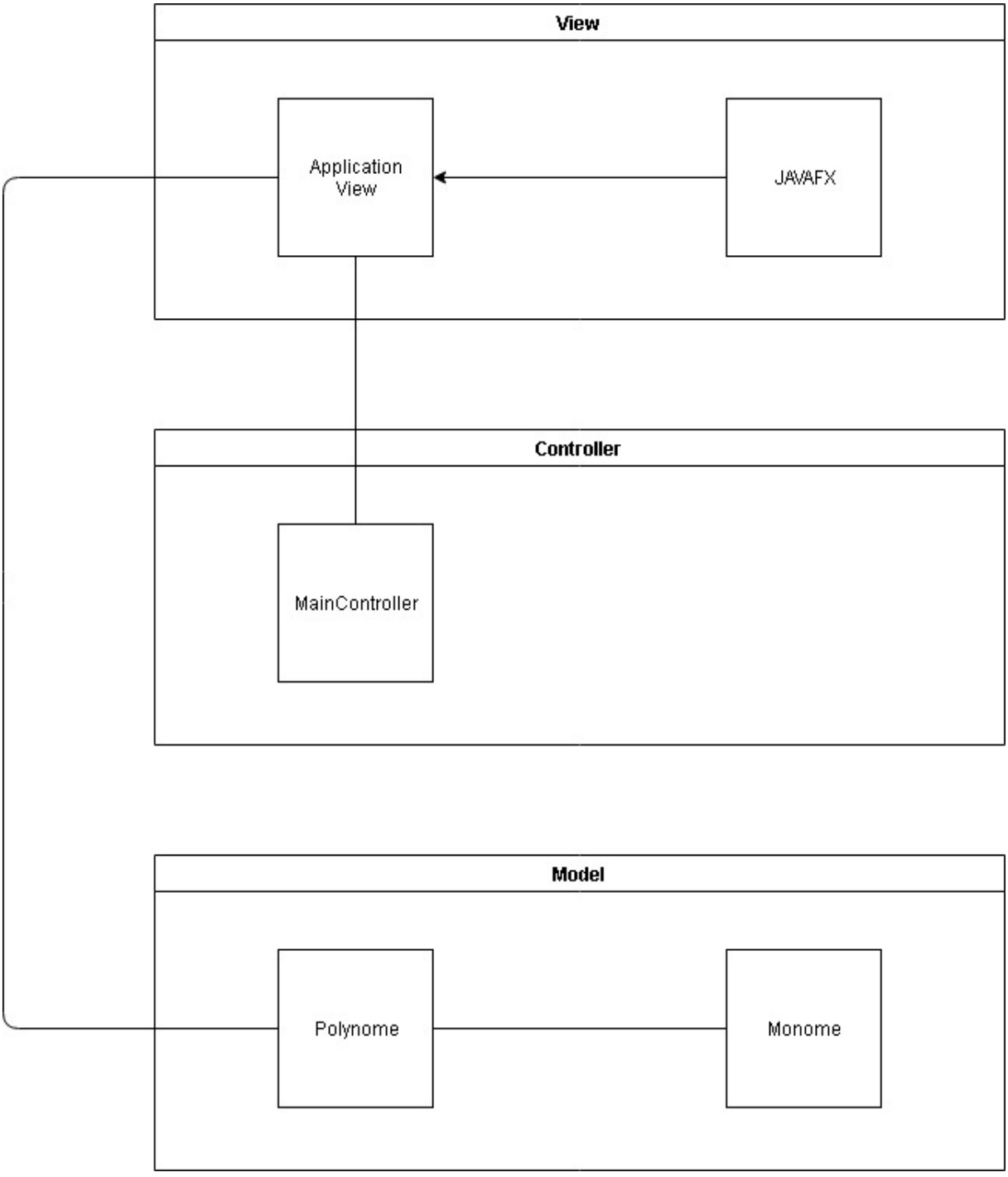
- Rezultatul oricărei operații este afișat utilizatorului prin intermediul interfeței grafice, împreuna cu operandul sau operanzii introduși.

Procesul de obținere al rezultatului droit prin intermediul aplicației dezvoltate este prezentat si in următorul *„ Flow Chart ” :*



4 PROIECTARE

După cum este specificat in cerința proiectului, acesta este organizat după structura MVC (Model-View-Controller). Astfel, pachetele folosite poarta numele acestei structuri, ar diagrama UML a acestora este prezentata mai jos :



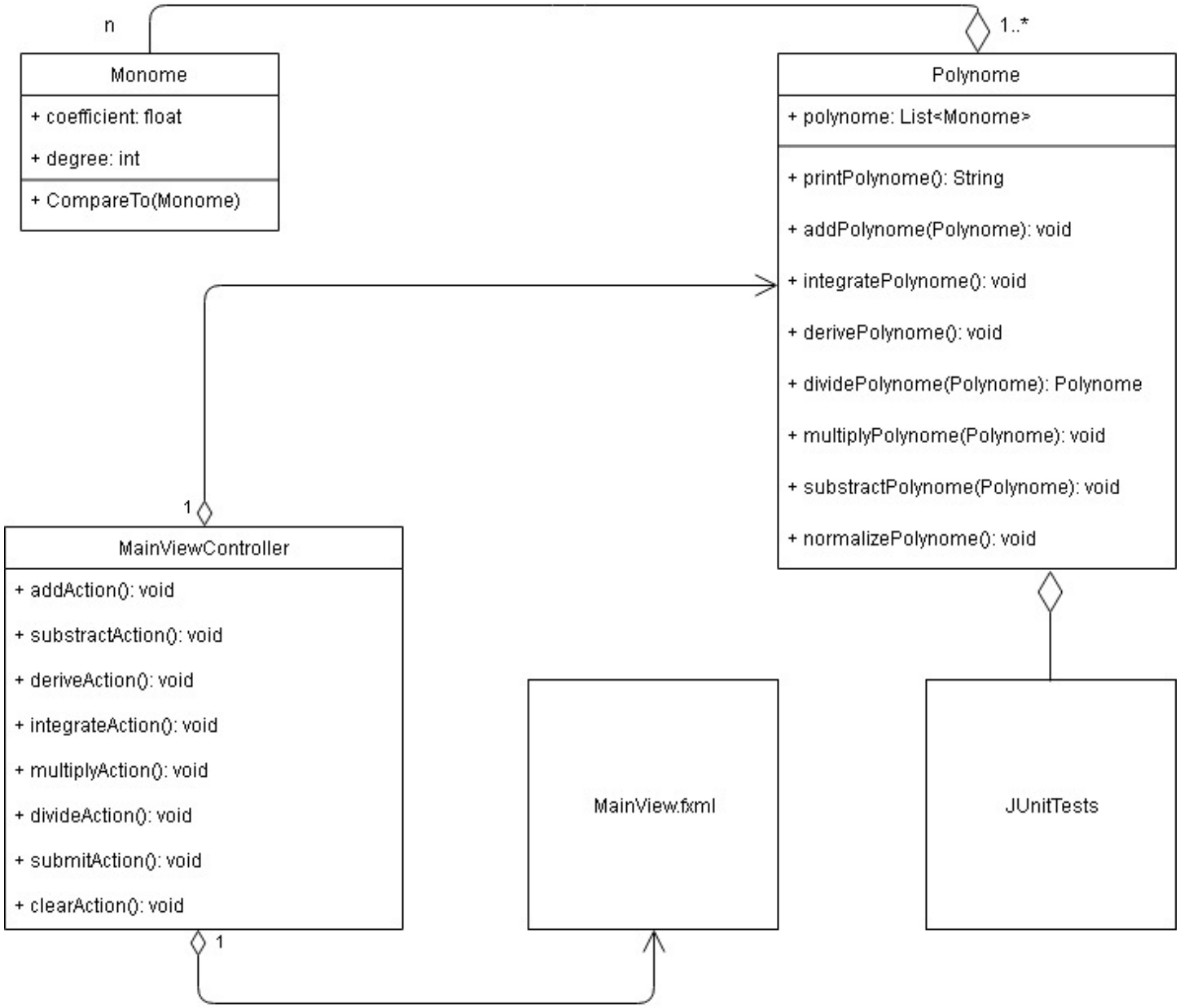
Urmând structura prezentata, am dezvoltat aplicația prin intermediul următoarelor clase Java:

- Monome;
- Polynome;
- MainViewController;
- JUnitTests;

Structura de care sta la baza proiectului este reprezentata de clasa Monome.java. Aceasta este utilizata de către clasa Polynome.java , care conține o lista de monoame, reprezentând o expresie polinomiala.

Casa MainViewController.java este cea care furnizează comportamentul interfeței, interfață care a fost implementata folosind JavaFX, respectiv un fișier .FXML, căruia i-a fost atribuit designul prin intermediul limbajului CSS, respectiv al fișierului style.css.

Diagrama UML de mai jos prezinta relațiile mai sus descrise :



5 IMPLEMENTARE

5.1 CLASA MONOME.JAVA

```
package model;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Monome implements Comparable<Monome>{
    int degree;
    float coefficient;

    public Monome(int degree, float coefficient) {. . .}

    public Monome(String monome) {. . .}

    public int getDegree() {. . .}

    public void setDegree(int degree) {. . .}

    public float getCoefficient() {. . .}

    public void setCoefficient(float coefficient) {. . .}

    @Override
    public int compareTo(Monome o) {. . .}
}
```

Clasa Monome, structura de baza a aplicației, retine doua variabile:

→ degree
→ coefficient

Aceste doua variabile sunt de tip int respectiv float. Tipul float este utilizat deoarece putem ajunge la coeficienți reali după efectuarea anumitor operații , precum derivare, integrare sau împărțire.

Ca si metode, am implementat *getters* si *setters* pentru a putea controla valoarea variabilelor definite, chiar daca acest lucru nu era obligatoriu deoarece variabilele sunt declarate implicit ca *public*.

Pe lângă acestea, metoda *compareTo* trebuit suprascrisa pentru a putea fi implementata funcția de sortare din cadrul clasei Polynome. Metoda returnează o valoare mai mica, egala sau mai mare decât zero in vederea sortării in ordine descrescătoare in funcție de gradul monoamelor.

5.2 CLASA POLYNOME.JAVA

```
package model;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Polynome {
    List<Monome> polynome = new ArrayList<Monome>();

    public Polynome(List<Monome> polynome) {. . .}

    public Polynome(String polynome) throws Exception {. . .}

    public List<Monome> getPolynome() {. . .}

    public void setPolynome(List<Monome> polynome) {. . .}

    public void normalizePolynome() throws Exception {. . .}

    private boolean isPolynome(String polynome){. . .}

    private Monome hasDegree(Monome monome){. . .}

    public  String printPolynome(){. . .}

    public void addPolynome(Polynome polynome) throws Exception {. . .}

    public void substractPolynome(Polynome polynome) throws Exception {. . .}

    public void multiplyPolynome(Polynome polynome){. . .}

    public Polynome dividePolynome(Polynome polynome) throws Exception {. . .}

    public void integratePolynome(){. . .}

    public void derivePolynome(){. . .}
}
```

Clasa Polynome este dependenta de clasa Monome si are ca variabila o lista de monoame. Astfel, am implementat doi constructori. Constructorul de baza are ca parametru o lista de monoame. Celălalt constructor este cel pe care l-am utilizat in implementarea propriu zisă a funcționalității, si are ca parametru un sir de caractere. Acest sir de caractere este cel introdus de către utilizator in câmpul destinat din interfața. Folosind funcțiile Regex, constructorul verifica inițial daca șirul este alcătuit doar din polinoame de forma : „ $\pm a x \hat{} b$ ”. Acestea sunt transmise constructorului cu parametru String din clasa Monome, unde la rândul lor sunt extrași coeficienții si gradele. Astfel se construiește o lista de monoame care este atribuita variabilei polynome declarate.

Metodele implementate vizează acoperirea cerințelor proiectului, respectiv efectuarea de operații asupra structurii de date descrise.

Metoda de normalizare este una cruciala in funcționarea corecta a tuturor celorlalte funcții . Aceasta are ca scop ordonarea descrescătoare a listei de monoame si eliminarea tuturor monoamelor cu coeficient zero. Aceasta este apelata atât in constructorii definiți cat si la finalul fiecărei operații.

Metoda de adunare primește ca parametru un alt polinom care este adăugat termen cu termen la cel din partea căruia sa făcut apelul funcției. Acest lucru se realizează prin parcurgerea polinomului de adăugat prin intermediul unui *foreach*. In cazul in care, polinomul asupra căruia se face adunarea, conține un monom de gradul celui parcurs, coeficienții acestora se aduna, iar in caz contrar monomul este adăugat in lista polinomului supus operației de adunare.

Metoda de scădere este similara celei de adunare cu diferența ca coeficienții monoamelor polinomului sustras sunt negate înainte de a fi adăugate celuilalt polinom.

Funcția de înmulțire are ca parametru tot al doilea membru al operației. Aceasta a fost implementata folosind doua structuri de foreach, cate una pentru fiecare polinom, cu importanta precizare ca un for se afla in interiorul celuilalt. Astfel, fiecare element al polinomului transmis ca parametru este înmulțit cu fiecare monom al celuilalt. Înmulțirea monoamelor se realizează prin înmulțirea coeficienților si adunarea gradelor intre ele. La finalul metodei polinomul este normalizat.

Metoda de împărțire urmează îndeaproape modelul matematic de divizare a polinoamelor lungi de coeficienți întregi si de o singura variabila. Aceasta presupune împărțirea primului monom al împărțitorului cu respectivul monom al deîmpărțitului. Rezultatul este reținut într-un polinom din cadrul funcției si, totodată înmulțit cu fiecare monom al împărțitorului. Polinomul rezultat in urma acestei înmulțiri este sustras din polinomul asupra căruia este aplicata împărțirea. Procesul continua pana in momentul in care polinomul rezultat in urma ultimei operații de scădere este gol, deoarece după fiecare set de operații polinomul este normalizat, ceea ce duce la eliminarea monoamelor cu coeficient nul, sau in momentul in care gradul aceluiași polinom este mai mic decât cel al împărțitorului. In cazul din urma, ceea ce a mai rămas din polinomul inițial este returnat de către metoda într-un nou polinom cu valoare de rest al împărțirii.

Metoda de derivare este de tip *void* fără parametrii. Aceasta deriveaza pe rând fiecare monom al polinomului, aceștia fiind parcurși prin intermediul unui

foreach. Derivarea se realizează matematic, si anume: gradul polinomului este înmulțit cu coeficientul acestuia urmând ca apoi sa fie decrementat.

Integrarea polinoamelor se realizează după modelul derivării, cu diferența ca inițial gradul monomului este incrementat urmând sa fie folosit pentru înmulțirea cu coeficientul.

Funcția de afișare din cadrul clasei Polynome este una complexa. Puteam opta pentru o afișare simplificata din punct de vedere al complexității metodei însă am optat pentru simplificarea interacțiunii utilizatorului cu aplicația. Astfel, funcția tine cont de valoarea coeficientului si al gradului fiecărui monom di polinom, iar in cazul in care aceștia au valoarea 1 sau 0, se elimina caracterele suplimentare, inutile din stringul generat. In cazul polinoamelor fără elemente funcția generează stringul „0”.

CLASA Polynome mai conține unele metode private, folosite in cadrul celorlalte funcții, pentru a simplifica sintaxa codului.

5.3 CLASA MAINVIEWCONTROLLER.JAVA

```
package controller;
import ...

public class MainViewController implements Initializable {

    Polynome firstPolynome, secondPolynome = null;

    @FXML
    private TextField input;

    @FXML
    private TextArea display;

    @FXML
    private Button add;

    @FXML
    private Button substract;

    @FXML
    private Button multiply;

    @FXML
    private Button divide;

    @FXML
    private Button derive;

    @FXML
    private Button integrate;

    @FXML
    private Button submit;

    @FXML
```

```
    private Button clear;

    @FXML
    void addAction(ActionEvent event) {. . .}

    @FXML
    void substractAction(ActionEvent event) {. . .}

    @FXML
    void deriveAction(ActionEvent event) {. . .}

    @FXML
    void integrateAction(ActionEvent event) {. . .}

    @FXML
    void multiplyAction(ActionEvent event) {. . .}

    @FXML
    void divideAction(ActionEvent event) {. . .}

    @FXML
    void submitAction(ActionEvent event) {. . .}

    @FXML
    void clearAction(ActionEvent event) {}

    public void initialize(URL location, ResourceBundle resources) {. . .}
}
```

Aceasta clasa furnizează comportamentul interfeței furnizate utilizatorului. Folosind JavaFX, controlerul prezinta adnotațiile @FXML deoarece interfața a fost descrisa într-un fișier .FXML .

Fiecare buton re un id specific căruia totodată ia fost atribuita si o metoda. Acest lucru nu era obligatoriu, puteam folosi o singura metoda unde sa verific ce buton a fost acționat, însă împărțirea acesteia in mai multe funcții am considerat-o necesara pentru un cod lizibil.

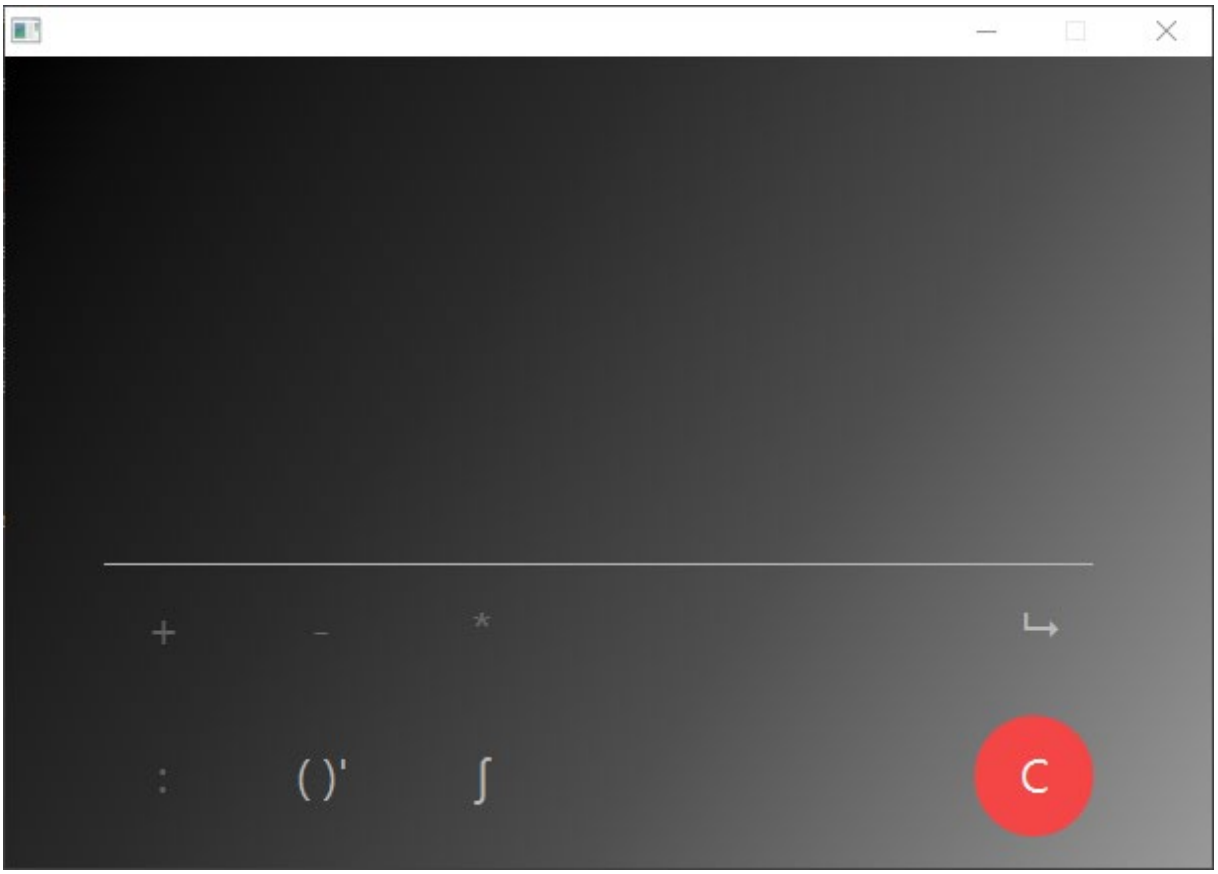
Metoda de inițializare a interfeței dezactivează butoanele de adunare, scădere, înmulțire si împărțire. Singurele butoane disponibile utilizatorului in momentul deschiderii ferestrei sunt cele pentru derivare, integrare de ștergere a memoriei (clear) si enter (pentru introducerea polinomului in memorie). La apăsarea butonului de derivare sau integrare, controllerul încearcă sa construiască un nou polinom din stringul care ia fost furnizat prin intermediul câmpului pus la dispoziția utilizatorului. In cazul in care acest lucru eșuează, respectiv constructorul arunca o excepție, câmpul de introducere a informațiilor se înroșește pentru ai sugera utilizatorului ca șirul de caractere introdus nu respecta sintaxa ceruta. In cazul in care șirul este acceptat, rezultatul operații alese se afișează pe ecran.

Butonul de introducere a informațiilor pentru un alt polinom inversează starea de dezactivare a butoanelor delegate cu efectuarea operațiilor aritmetice.

In cazul apăsării unuia dintre ele, procesul este același, iar după validarea polinomului rezultatul este transmis căsuței de afișare.

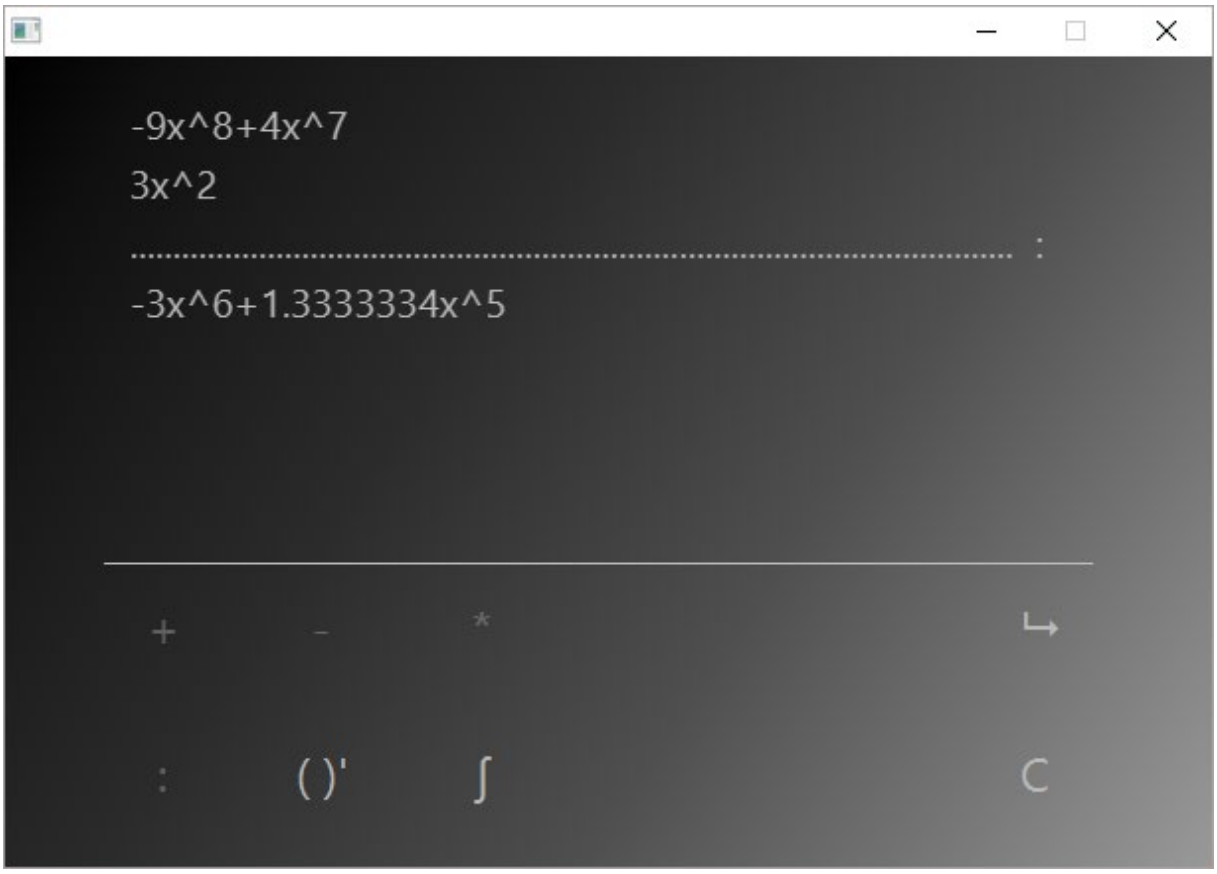
5.4 INTERFAȚA GRAFICĂ

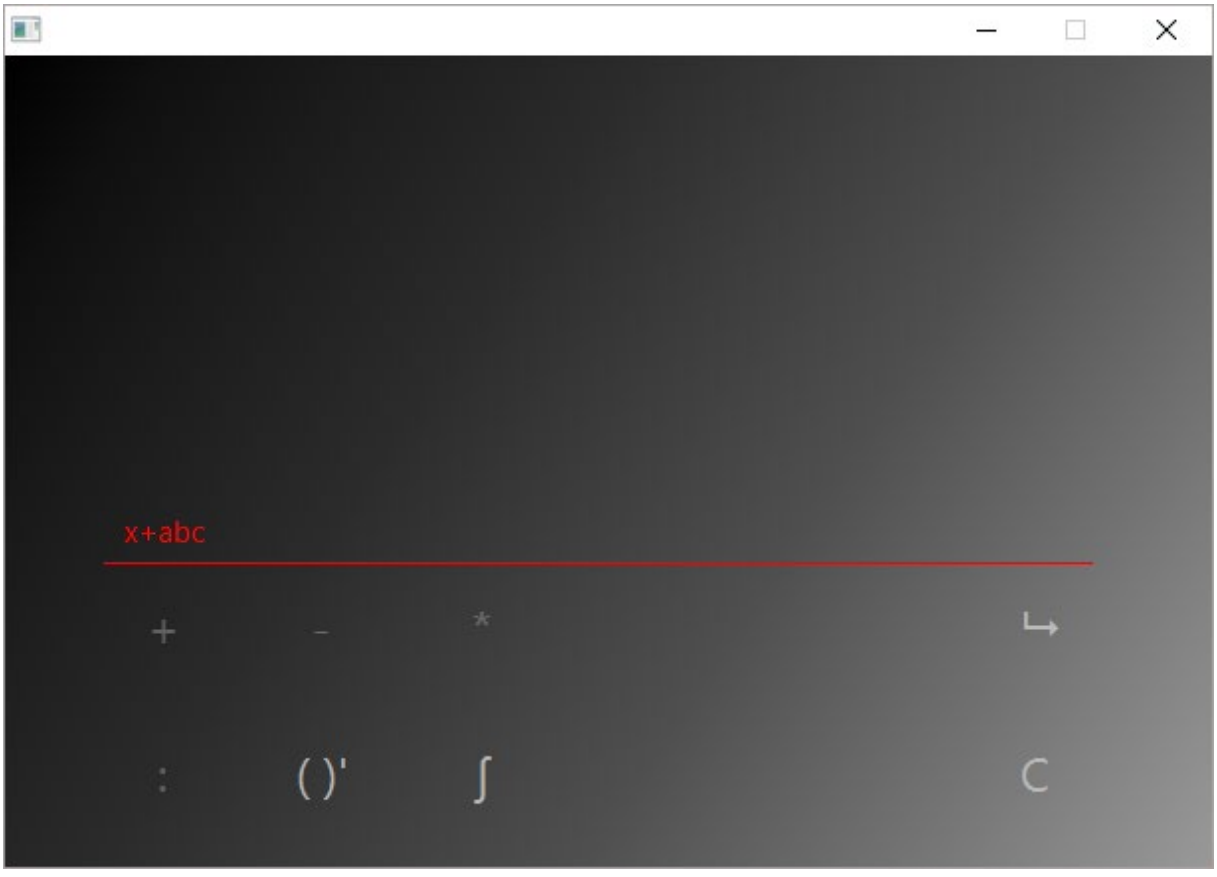
Utilizarea JavaFX mi-a permis crearea unei interfețe prietenoase cu utilizatorul, cu aspect modern. Acest lucru a fost realizat prin alegerea unui design modern dar in special prin utilizarea fișierului de CSS. Acesta a permis descrierea comportamentului variat al butoanelor din punct de vedere vizual, dar si utilizarea de culori spectaculoase.



Am pus accentul pe o implementare cat mai minimalista a interfeței, acest lucru fiind evident prin aspectul butoanelor, care sunt descrise du cate un singur caracter corespunzător operației pe care acesta o realizează.

Mesajele furnizate utilizatorului sunt evidente si sugestive, precum zona mare de afișare a operației efectuate dar si prin colorarea câmpului de introducere a datelor in momentul in care șirul introdus nu respecta sintaza dorita.


-9x^8+4x^7
3x^2
:
-3x^6+1.3333334x^5
+
*
:
( )'
∫
C


x+abc
+
*
:
( )'
∫
C

6 REZULTATE

Rezultatele implementării au fost testate prin intermediul JUnit4, într-o clasa separata unde am verificat funcționarea corecta a tuturor operațiilor si a construcției corecte a polinoamelor primite ca String.

```
import model.Polynome;
import org.junit.Assert;
import org.junit.Test;

public class JUnitTests {

    Polynome firstPolynome = new Polynome("34x^4-56x^3+4x^0");
    Polynome secondPolynome = new Polynome("-3x^3+6x^1");

    public JUnitTests() throws Exception {
    }

    @Test
    public  void inputTest() throws Exception {
        try{
            Polynome p = new Polynome("x");
            Assert.assertFalse(false);
        }
        catch (Exception e){
            Assert.assertTrue(true);
        }

        try{
            Polynome p = new Polynome("1x");
            Assert.assertFalse(false);
        }
        catch (Exception e){
            Assert.assertTrue(true);
        }

        try{
            Polynome p = new Polynome("1x^2");
            Assert.assertTrue(true);
        }
        catch (Exception e){
            Assert.assertFalse(false);
        }

        try{
            Polynome p = new Polynome("-3x");
            Assert.assertFalse(false);
        }
        catch (Exception e){
            Assert.assertTrue(true);
        }

        try{
            Polynome p = new Polynome("-3x^7");
            Assert.assertTrue(true);
        }
        catch (Exception e){
            Assert.assertFalse(false);
        }

        try{
            Polynome p = new Polynome("-67x^7+56x^7");
            Assert.assertTrue(true);
        }
        catch (Exception e){
```

```
            Assert.assertFalse(false);
        }
    }

    @Test
    public void addTest() throws Exception {
        firstPolynome.addPolynome(secondPolynome);
        Assert.assertEquals("34x^4-59x^3+6x^1+4", firstPolynome.printPolynome());
    }

    @Test
    public void substractTest() throws Exception {
        firstPolynome.substractPolynome(secondPolynome);
        Assert.assertEquals("34x^4-53x^3-6x^1+4", firstPolynome.printPolynome());
    }

    @Test
    public void multiplyTest(){
        firstPolynome.multiplyPolynome(secondPolynome);
        Assert.assertEquals("-102x^7+168x^6+204x^5-336x^4-12x^3+24x^1",
firstPolynome.printPolynome());
    }

    @Test
    public void divideTest() throws Exception {
        firstPolynome.dividePolynome(secondPolynome);
        Assert.assertEquals("-11.333333x^1+18.666666",
firstPolynome.printPolynome());
    }

    @Test
    public void deriveTest(){
        firstPolynome.derivePolynome();
        Assert.assertEquals("136x^3-168x^2", firstPolynome.printPolynome());
    }

    @Test
    public void integrateTest(){
        firstPolynome.integratePolynome();
        Assert.assertEquals("6.8x^5-14x^4+4x^1", firstPolynome.printPolynome());
    }
}
```

7 CONCLUZII

Aceasta tema a condus la dobândirea unor cunoștințe care nu pot fi însușite decât prin exercițiu, iar tema efectiv abordata este de importanta redusa comparativ cu copul educativ al cerinței. Scopul acestui proiect consta dup aparerea mea in învățarea prin greșeala si succes.

8 BIBLIOGRAFIE

- Peter H. Selby, Steve Slavin, Practical Algebra: A Self-Teaching Guide, 2nd Edition, Wiley, ISBN-10 0471530123 ISBN-13 978-0471530121
- https://www.draw.io/