Documentatie Tema 1

Calculator Polinoame

Iurieț Denis – Ionuț

Grupa: 30223

Profesor Laborator : Moldovan Dorin

Contents

[1.Obiectivul temei 3](#_Toc3674755)

[2.Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc3674756)

[2.1 Analiza problemei 3](#_Toc3674757)

[2.2 Modelare 4](#_Toc3674758)

[2.3 Scenarii 4](#_Toc3674759)

[2.4 Cazuri de utilizare 5](#_Toc3674760)

[3.Proiectare 5](#_Toc3674761)

[4.Implementare 6](#_Toc3674762)

[4.1 Clasa Monom 6](#_Toc3674763)

[4 .2 Clasa Polinoms 7](#_Toc3674764)

[4.3 Clasa View 9](#_Toc3674765)

[4.4 Clasa Controller 9](#_Toc3674766)

[5.Rezultate 11](#_Toc3674767)

[6.Concluzii 11](#_Toc3674768)

[7.Bibliografie 11](#_Toc3674769)

# 1.Obiectivul temei

Propunerea, proiectarea și implementarea unui sistem de procesare a polinoamelor de o singură variabilă cu coeficienți întregi.

Obiectivele secundare

|  |  |  |
| --- | --- | --- |
| Obiectiv secundar | Descriere | Capitol |
| Analiza problemei, modelare, scenarii, cazuri de utilizare | Prezentarea cadrului de cerințe funcționale formalizat și cazurile de utilizare ca și diagrame și descrieri de use-case. | 2 |
| Proiectare | Prezentarea proiectării OOP a aplicației, diagramele UML de clase, structurile de date folosite, interfețele definite și algoritmii folosiți. | 3 |
| Implementare | Descrierea claselor cu câmpurile și metodele importante și de asemenea, descrierea implentării interfeței utilizator. | 4 |
| Testare | Prezentarea scenariilor pentru testare, folosind framework-ul de testare JUnit. | 5 |

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

## 2.1 Analiza problemei

Proiectarea orientată pe obiecte funcționează în jurul entităților și a caracteristicilor lor în locul funcțiilor implicate în sistemul software.

Pentru analiza problemei, am pornit de la componentele specifice și de bază ale proiectului, metodă specifică modelului de design bottom-up. Această metodă continuă cu compunerea unui nivel mai ridicat de componente, utilizând componentele de bază sau de nivel inferior.

Procesul de proiectare a software-ului poate fi perceput ca o serie de etape bine definite. De exemplu, identificarea obiectelor și gruparea acestora în clase, definirea ierarhiei claselor și a relațiilor dintre acestea, pornind de la alegerea unor substantive și verbe, potrivite pentru numele claselor, respectiv a metodelor.

## 2.2 Modelare

Modelarea aplicației se bazează pe conceptele programării orientate pe obiect, și anume:

Obiect = componentă software care incorporează atât atributele cât şi operaţiile care se pot efectua asupra atributelor şi care suportă moştenirea;

Clase - o clasă este o descriere generalizată a unui obiect. Un obiect este o instanță a unei clase. Clasa definește toate atributele pe care un obiect le poate avea și metodele care definesc funcționalitatea obiectului;

Încapsulare – unește informațiile importante ale unui obiect, dar restricționează și accesul la date și metode din exterior;

Moștenirea - permite claselor similare să se suprapună în mod ierarhic, unde clasele inferioare sau subclasele pot importa, pune în aplicare și reutiliza variabilele și metodele permise din clasele superioare immediate;

Polimorfismul – abilitatea de a lua mai multe forme, aceeași metodă folosită într-o superclasă poate fi suprascrisă în subclase pentru a da o funcționalitate diferită.

## 2.3 Scenarii

Folosirea corespunzătoare a aplicației presupune introducerea în câmpurile “Coefficient”, respectiv “Exponent” și apăsarea butonului “Enter” pentru a se finaliza adăugarea monomului la polinom. Acest lucru trebuie repetat în cazul în care se dorește realizarea operațiilor de adunare, scădere sau înmulțire, operații ce necesită două polinoame.

Există probabilitatea ca un utilizator să utilizeze aplicația într-un mod neadecvat, ceea ce conduce la posibile situații nedorite, cum ar fi:

-Introducerea de caractere sau string-uri în câmpurile destinate adăugării de elemente;

-Introducerea de valori necorespunzătoare atributelor monomului;

-Introducerea directă a polinomului în câmpul adresat afișării rezultatului;

-Introducerea celui de-al doilea polinom, chiar dacă se realizează o operație ce efectuează modificări doar asupra primului polinom.

## 2.4 Cazuri de utilizare

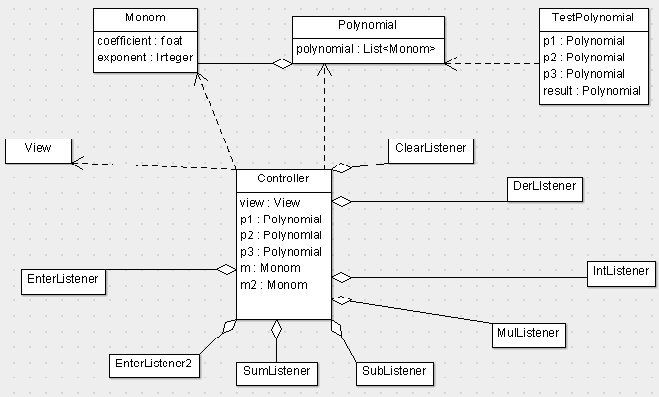
Funcționalitățile oferite de către calculatorul polinomial sunt:

-Operațiile de adunare, scădere, înmulțire între primul polinom și cel de al doilea;

-Derivarea și integrarea primului polinom;

-Sortarea polinoamelor în ordine descrescătoare a gradului monomului, după fiecare inserare și simplificare a termenilor.

# 3.Proiectare



Aplicația “Polynomial calculator” are la baza 5 clase și este organizată conform șablonului Model-View-Controller (MVC). Acest tip de organizare reprezintă separarea interfeței utilizator într-o veder(View) care interacționează cu Modelul după nevoi, și un Controller care răspunde la cererile utilizatorului, interacționând atât cu View-ul, cât și cu Modelul.

Clasa Monom – reprezintă clasa ce operează asupra elementelor unui polinom, având ca atribute un coeficient și un exponent. De asemenea, aceasta conține metode de prelucrare a monoamelor precum adunarea, scaderea, inmultirea, integrarea și derivarea, precum și metoda toString() folosită pentru afișarea acestora;

Clasa Polinom – realizează operațiile enumerate mai sus în cadrul clasei “Monom”. Această clasă are în structura sa o metodă pentru aranjarea polinoamelor ce reprezintă o ordonare a termenilor in ordine descrescătoare din punct de vedere al gradului acestora și a simplificării termenilor în cazul în care gradul a două sau mai multe monoame coincide. În această clasă, am utilizat colecția de tip “List” pentru reprezentarea polinoamelor și stocare monoamelor.

Clasa View – în cadrul acestei clase, ce realizează interfața utilizator;

Clasa Controller – aceasta tratează interacțiunea utilizatorului folosind ascultatori, implementați sub forma unor clase interne. Apelează vizualizarea și modelul după nevoi.

Clasa MainClass – are rolul de a instanția celelalte clase.

# 

# 4.Implementare

## 4.1 Clasa Monom

Implementează interfața Comparable pentru sortarea polinoamelor în ordine descrescătoare după gradul monoamelor și are ca atribute coeficientul de tip float și exponentul de tip int.

**public** **class** Monom **implements** Comparable<Monom> {

**private** **float** coefficient;

**private** **int** exponent;

**public** Monom(**float** c, **int** e) {

**this**.coefficient = c;

**this**.exponent = e;

}

**public** Monom(Monom m) {

**this**.coefficient = m.coefficient;

**this**.exponent = m.exponent;

}

**public** String toString() {

**if** (**this**.exponent != 0 && **this**.coefficient > 0) {

**return** "+" + **this**.coefficient + "X" + "^" + **this**.exponent;

} **else** **if** (**this**.exponent != 0 && **this**.coefficient < 0) {

**return** **this**.coefficient + "X" + "^" + **this**.exponent;

} **else** **if** (**this**.exponent == 0 && **this**.coefficient > 0) {

**return** "+" + **this**.coefficient;

} **else** **if** (**this**.exponent == 0 && **this**.coefficient < 0)

{

**return** " " + **this**.coefficient;

} **else** {

**return** " ";

}

}

Metoda toString() este utilizată la afișarea polinoamelor. Aceasta are mai multe ramuri if pentru diferite cazuri, afișarea unui monom fiind influențată de valoarea coeficientului sau exponentului acestuia.

**public** **int** compareTo(Monom m) {

**return** (**int**) (m.getExponent()) - **this**.exponent;

}

Această metoda este folosită pentru implementarea interfeței “Comparable”.

## 4 .2 Clasa Polynomial

**public** **class** Polynomial {

**private** List<Monom> polynomial = **new** ArrayList<Monom>();

Are în structura sa, o colecție de tip “List” ce permite manevrarea polinomului și stocarea monoamelor.

**public** **void** addElements(Monom x) {

Monom m = **new** Monom(x.getCoefficient(), x.getExponent());

**this**.polynomial.add(m);

}

“addElements” este o metodă cu care se realizează adăugarea unui nou monom la polinom.

**public** **void** addPolynomials(Polynomial p1, Polynomial p2) {

**int** i = 0, j = 0, k = 0;

**while** (i < p1.getPolynomial().size() && j < p2.getPolynomial().size()) {

**if** (p1.getPolynomial().get(i).getExponent() < p2.getPolynomial().get(j).getExponent()) {

**this**.addElements(p2.getPolynomial().get(j));

k++;

j++;

} **else** **if** (p1.getPolynomial().get(i).getExponent() > p2.getPolynomial().get(j).getExponent()) {

**this**.addElements(p1.getPolynomial().get(i));

k++;

i++;

} **else** **if** (p1.getPolynomial().get(i).getExponent() == p2.getPolynomial().get(j).getExponent()) {

**this**.addElements(p1.getPolynomial().get(i));

i++;

**this**.polynomial.get(k).addMonoms(p2.getPolynomial().get(j));

k++;

j++;

}

}

**while** (i < p1.getPolynomial().size()) {

**this**.addElements(p1.getPolynomial().get(i));

i++;

}

**while** (j < p2.getPolynomial().size()) {

**this**.addElements(p2.getPolynomial().get(j));

j++;

}

**this**.sortPolynomial();

}

Implementarea metodei “addPolynoms” este bazată pe algoritmul de interclasare. Se verifică dacă elementul din primul polinom are gradul mai mare decât cel din al doilea polinom, în cazul in care din if rezultă valoarea logică ‘1’, atunci elementul din primul polinom va fi adăugat in polinomul rezultat, dacă valoarea logică este ‘0’, atunci se va trece la urmatoarea structură if care verifică dacă gradul elementului din polinomul 2 este mai mare. În cazul în care răspunsurile celor două verificări de mai sus sunt negative, atunci exponenții sunt egali și cu ajutorul metodei “addMonoms”, cele două elemente vor fi însumate, monomul rezultat fiind adaugat în polinomul final.

Dacă un polinom conține mai multe elemente, atunci cu ajutorul structurilor “while” de la finalul metodei, elementele neparcurse, vor fi adăugate.

**public** **void** subPolynomials(Polynomial p1, Polynomial p2) {

**for** (Monom index2 : p2.getPolynomial()) {

index2.negative();

}

**this**.addPolynomials(p1, p2);

}

Metoda folosită pentru scăderea polinoamelor, este asemănătoare cu cea de adunare, diferența fiind că elementele polinomului 2 vor fi înmulțite cu -1.

**public** **void** mulPolynomials(Polynomial p1, Polynomial p2) {

Monom m3 = **new** Monom();

**for** (Monom index : p1.getPolynomial()) {

m3 = **new** Monom(index);

**for** (Monom index2 : p2.getPolynomial()) {

m3.mulMonoms(index2);

**this**.addElements(m3);

m3 = **new** Monom(index);

}

}

**this**.sortPolynomial();

}

**public** **void** derivatePolynomial(Polynomial p1) {

**for** (Monom index : p1.getPolynomial()) {

Monom m = **new** Monom(index);

m.derivateMonom();

**if**(m.getCoefficient() != 0)

**this**.polynomial.add(m);

}

}

**public** **void** integratePolynomial(Polynomial p1) {

**for** (Monom index : p1.getPolynomial()) {

Monom m = **new** Monom(index);

m.integrateMonom();

**this**.polynomial.add(m);

}

}Metodele de derivare și integrare se realizează cu o parcurgere a polinomului și apoi se apeleaza metodele din clasa Monom.

**public** **void** integrateMonom() {

**this**.exponent = **this**.exponent + 1;

**this**.coefficient = **this**.coefficient / **this**.exponent;

}

**public** **void** derivateMonom() {

**if** (**this**.exponent != 0) {

**this**.coefficient = **this**.coefficient \* **this**.exponent;

**this**.exponent = **this**.exponent - 1;

} **else** {

**this**.coefficient = 0;

}

}

## 4.3 Clasa View

**public** **class** View **extends** JFrame {

**public** View() {

**this**.setBounds(100, 100, 650, 589);

**this**.setVisible(**true**);

**this**.setTitle("Polynomial calculator");

**this**.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

La acest frame, am adăugat un singur panel al cărui layout l-am setat “absolute/null” pentru a așeza elementele grafice după bunul plac.

panel.setBounds(0, 0, 632, 542);

**this**.add(panel);

panel.setLayout(**null**);

În această clasă s-au declarat toate butoanele, label-urile, textfield-urile și s-au așezat în panel.

## 4.4 Clasa Controller

**private** View view;

**private** Polynomial polynom = **new** Polynomial();

**private** Polynomial polynom2 = **new** Polynomial();

**private** Polynomial polynom3 = **new** Polynomial();

**private** Monom monom;

**private** Monom monom2;

S-au declarat elemente ale modelului și ale vederii. Vom folosi polinoamele, pentru citirea lor în interfață și pentru afișare.

**class** EnterListener **implements** ActionListener {

**public** **void** actionPerformed(ActionEvent e) {

**int** coefficient = 0;

**int** exponent = 0;

**try** {

coefficient=Integer.*parseInt*(view.getCoefficientTf().getText());

exponent=Integer.*parseInt*(view.getExponentTf().getText());

monom = **new** Monom(coefficient, exponent);

polynom.addElements(monom);

polynom.sortPolynom();

view.getPolynomTf().setText(polynom.printPolynoms());

view.getCoefficientTf().setText("");

view.getExponentTf().setText("");

} **catch** (NumberFormatException nfex) {

JOptionPane.*showMessageDialog*(view.getPanel(), "Bad input!");

}

}

}

Pentru fiecare buton s-a creat o clasă internă pentru gestionarea ascultătorilor. Aceasta este folosită pentru butonul enter care creează primul polinom.Variabilele “coefficient” și “exponent” preiau valorile introduse în text field-urile corespunzătoare, cu ajutorul cărora se vor crea elementele ce vor fi introduse în polinom. La fiecare adăugare, va avea loc o sortare a polinomului și apoi afișarea acestuia. În cazul unei introduceri eronate, se va afișa un mesaj “Bad input !” și se va cere o nouă introducere a datelor.

Pentru cel de-al doilea polinom, clasa este identică.

**class** SumListener **implements** ActionListener {

**public** **void** actionPerformed(ActionEvent e) {

view.getResultTf().setText("");

polynom3 = **new** Polynomial();

polynom3.addPolynomials(polynom, polynom2);

polynom3.sortPolynomial();

view.getResultTf().setText(polynom3.printPolynomial());

}

}

La apăsarea butonului “Add”, se vor însuma cele două polinoame înscrise. Aceasta este clasa care implementează ascultatorul pentru butonul de sumă. Se face o apelare a metodei “addPolynoms” din clasa “Polinoms”, o sortare a polinomului rezultat, iar apoi afișarea acestuia. Pentru toate cele cinci operații implementate, clasele sunt asemănatoare, însă diferă doar metoda apelată.

**class** ClearListener **implements** ActionListener {

**public** **void** actionPerformed(ActionEvent e) {

view.getResultTf().setText("");

view.getPolynomTf().setText("");

view.getPolynomTf2().setText("");

polynom = **new** Polynomial();

polynom2 = **new** Polynomial();

polynom3 = **new** Polynomial();

}

}

Acesta este un ascultător pentru butonul “Clear”. La apăsarea acestui buton, se vor crea noi polinoame și se vor elibera toate text field-urile.

# 5.Rezultate

**public** **class** TestPolynomial {

Polynomial p1;

Polynomial p2;

Polynomial p3;

Polynomial result;

Am creat o clasă destinată testării operațiilor implementate, în care am declarat patru polinoame pentru a efectua testele.

@Test

**public** **void** testSum() {

p1 = **new** Polynomial();

p2 = **new** Polynomial();

p3 = **new** Polynomial();

result = **new** Polynomial();

//(2x^3 - 3X^2 + X + 4)

p1.addPolyElements(2, 3);

p1.addPolyElements(1, 1);

p1.addPolyElements(-3, 2);

p1.addPolyElements(4, 0);

//(5x^5 + X^4 + 4X^3 + 2X + 3)

p2.addPolyElements(5, 5);

p2.addPolyElements(1, 4);

p2.addPolyElements(2, 1);

p2.addPolyElements(3, 0);

p2.addPolyElements(4, 3);

//(5X^5 + X^4 + 6X^3 - 3X^2 + 3X + 7)

result.addPolyElements(5, 5);

result.addPolyElements(1, 4);

result.addPolyElements(6, 3);

result.addPolyElements(-3, 2);

result.addPolyElements(3, 1);

result.addPolyElements(7, 0);

result.sortPolynomial();

p3.addPolynomials(p1, p2);

*assertEquals*(p3.printPolynomial(), result.printPolynomial());

}

Testul pentru operația de adunare. În “p1” și în “p2” am adăugat polinoamele ce urmează a fi adunate, iar în “p3” am stocat rezultatul adunării. În “result” am pus rezultatul adunării pe care îl așteptăm.

Pentru toate operațiile testul se efectueză similar, diferențe posibile fiind metoda apelată sau numărul de polinoame folosite.

# 6.Concluzii

În concluzie, proiectul realizat îndeplinește cerințele oferite, cu excepția realizării operației de împărțire a două polinoame.

Din această primă temă, aș putea afirma că am învățat importanța organizării timpului acordat unui proiect, structurarea și analizarea cu atenție a problemei înainte de începerea implementării acesteia în vederea găsirii unei rezolvări eficiente.

Posibile dezvoltări ulterioare pot fi considerate: îmbunătățirea interfeței din punct de vedere al aspectului, realizarea aplicației astfel încât să permită introducerea directă a polinomului fără a fi necesară introducerea succesivă a monoamelor, afișarea corectă a puterilor fără folosirea semnului ‘^’, utilizând “Unicode subscripts and superscripts”. O altă posibilă imbunătățire ar fi eficientizarea unor metode.

# 7.Bibliografie

1. <https://stackoverflow.com/>

2. https://www.tutorialspoint.com/