***DOCUMENTAȚIE***

TEMA 1 TEHNICI DE PROGRAMARE

***Sistem de procesare a polinoamelor***

*Pogăcean Rahela- Alexandra,*

Calculatoare și Tehnologia Informației

Anul II, seria B, grupa 30226

**Cuprins**

1. **Obiective**
   1. **Obiectiv principal**
   2. **Obiective secundare**
2. **Analiza problemei**
   1. **Descriere**
   2. **Scenarii și cazuri de utilizare**
3. **Proiectare**
   1. **Decizii de proiectare**
   2. **Diagrame UML**
   3. **Structuri de date**
   4. **Proiectare clase și algoritmi**
4. **Implementare**
   1. **Metode**
   2. **Interfața utilizator**
5. **Rezultate și concluzii**
6. **Bibliografie**
7. **Obiective**
   1. **Obiectivul principal**

Obiectivul fundamental al acestei lucrări de laborator este propunerea, proiectarea și implementarea unui sistem de procesare a polinoamelor de o singură variabilă cu coeficienți întregi.

* 1. **Obiective secundare**

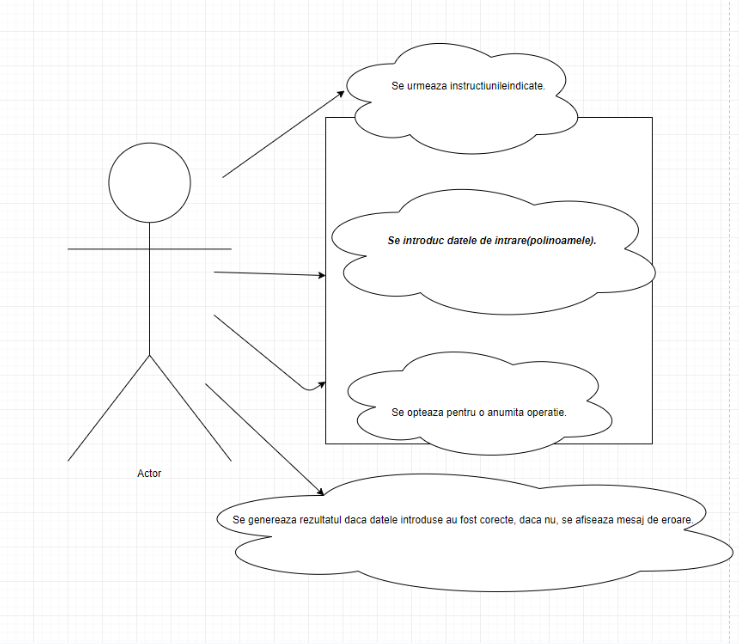
Un prim obiectiv secundar este constituit de respectarea paradigmelor programării orientate pe obiecte, structurarea codului in clase, ceea ce face ca acesta să fie mai lizibil și mai facil de urmărit. Realizarea diagramelor de clase și a use case-urilor reprezintă un alt obiectiv secundar al acestui proiect, precum și dezvoltarea ulterioară, propunându-se o serie de îmbunătățiri ale aplicației pentru o versiune mai complexă.

1. **Analiza problemei**

**2.1 Descriere**

Problema a fost analizată și evaluată pornind de la conceptul matematic care o definește. În domeniul matematicii, un polinom este o expresie alcătuită prin înlănțuirea mai multor monoame. Un monom este un termen format dintr-o constantă(numită coefficient) înmulțită cu una sau mai multe variabile. Fiecare variabilă poate avea un exponent constant întreg pozitiv. Un monom fără variabile se numește constant sau constantă, gradul unui termen constant fiind 0.

* 1. **Scenarii și cazuri de utilizare**



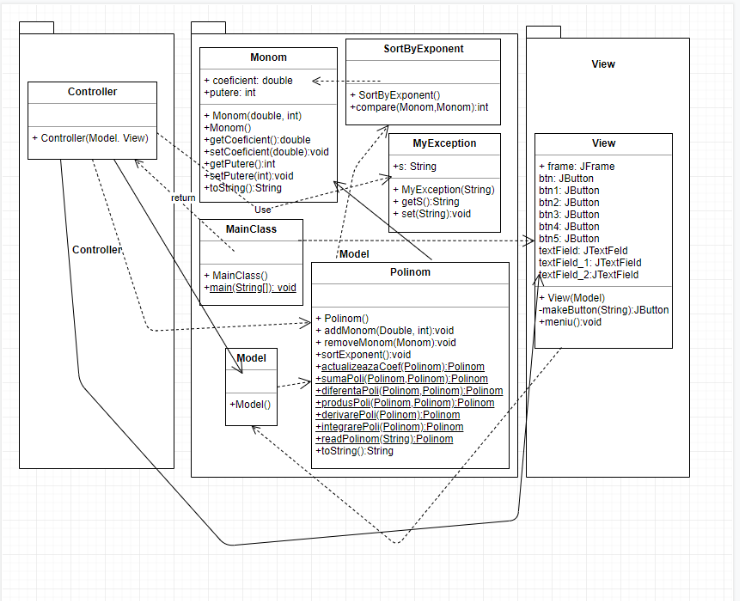
Folosirea calculatorului de polinoame presupune urmarea succesivă a mai multor etape. Astfel, prima dată se citesc indicațiile, sugestiile oferite, apoi se realizează introducerea datelor de la tastatura de către utilizator în casuțele de TextField. În cazul introducerii unor date valide, se continuă cu alegerea operației dorite prin apăsarea butonului specific și listarea rezultatului într-un alt câmp de text, iar dacă input-ul nu a respectat condițiile impuse, se va semnala un mesaj de alertă. Apăsarea butonului de resetare va conduce la o inițializare, curățare a câmpurilor destinate textului, fiind posibilă o nouă sesiune de folosire a calculatorului.

1. **Proiectare**

**3.1 Decizii de proiectare**

Am optat pentru divizarea acestei teme în mai multe task-uri. Pentru început, am decis să proiectez interfața grafică. Am testat funcționalitatea acesteia prin introducerea datelor de intrare (caractere) și apoi am afișat acele caractere în spațiile destinate răspunsului. Proiectarea interfeței grafice, asigurându-mă de corectitudinea datelor introduse și de afișarea corectă a lor a fost urmată de implementarea operațiilor cu polinoame (adunarea, scăderea, înmulțirea, derivarea și integrarea). Implementarea acestora a avut la bază aplicarea conceptelor și metodelor matematice, întreaga logică combinațională a proiectului avându-și rădăcinile în cunoștințele și experiența dobândite la această disciplină. Acest lucru constituie încă o dovadă temeinică a relației de interdependență dintre știința cifrelor și cea a informaticii, precum și a capacității de aplicare a uneia în cadrul celeilalte.

* 1. **Diagrame UML**



Unified Modeling Language(UML) este un limbaj standard pentru descrierea de modele si specificatii software. UML oferă o largă gamă de diagrame pentru modelarea diferitelor situații în cadrul unui proiect de dezvoltare software.

Aceste diagrame sunt de mai multe tipuri: de activitate, de componente, de clasă, package, de secvență, use-case și deployment.

În continuare, va fi prezentată diagrama de clase și de pachete pentru sistemul de procesare a polinoamelor implementat.

3.**3 Structuri de date**

Ca structuri de date, din categoria Collections am folosit ArrayList, implementat ca o secvență de valori stocate într-un tablou re-dimensionabil. Această listă a fost utilizată în clasa Monom, cu scopul de a stoca mai multe monoame, în ideea separării polinomului în mai mulți termeni (monoame).

* 1. **Proiectare clase**

1. **Implementare**
   1. **Metode**
   2. **Interfața utilizator**

În cele ce urmează, voi prezenta proiectarea claselor, algoritmii folosiți, implementarea metodelor și a interfeței utilizator.

Întregul proiect este structurat în model MVC (Model- View- Controller), care este un model arhitectural utilizat în ingineria software. Succesul modelului se datorează izolării logicii față de interfața cu utilizatorul, rezultând o aplicație modularizată, ușor de înțeles și eficientă. Aplicația este structurată în mai multe pachete, câte unul pentru fiecare funcționalitate (model, view, controller). Acestea, la rândul lor, conțin câte o clasă cu aceeași denumire. Modelul manipulează operațiunile logice și de utilizare de informație, constând în instanțierea a două obiecte de tipul polinom pe care se vor aplica operațiile menționate anterior. Lăsând la o parte abordarea de ansamblu, superficială și încercând pătrunderea în adevărata logică a programului, prin descoperirea gândirii din spatele acestei aplicații, gândire ce a guvernat, de fapt, întreg mecanismul, se observă structurarea pe nivele. Astfel, pentru a crea funcționalități complexe, s-a pornit de la lucruri și concepte simple, de bază, care au fost ulterior dezvoltate, interconectate și interclasate cu scopul de a obține un tot unitar. Astfel, clasa care a constituit punctul de plecare este Monom, având ca atribute coeficientul real și exponentul întreg. Aceste 2 proprietăți sunt necesare în ceea ce privește procesarea monoamelor și, ulterior, a polinoamelor, deoarece ele definesc acest concept matematic și fac posibilă manipularea lui. Clasa Monom conține constructorul ce are ca parametri cele 2 atribute amintite, getters și setters ce permit accesarea, respectiv setarea acestora, necesare întrucât acestea au fost declarate ca fiind private. De asemenea, mai este conținută metoda toString(), ce permite afișarea monomului în formatul dorit.

Având în vedere faptul că mai multe monoame înlănțuite formează un polinom, în urma implementării clasei Monom am continuat cu implementarea clasei Polinom, aplicând, într-o anumită măsură, abordarea Bottom-Up. Astfel, clasa Polinom cuprinde un ArrayList de monoame, instanțiat în constructorul fără parametri al clasei. Din necesitatea de a adăuga, respectiv a șterge monoame din listă, am implementat metodele de tip void addMonom(Double coef, int putere), respectiv removeMonom(Monom m), care facilitează aceste operații pe liste. O altă necesitate a fost ordonarea elementelor polinomului după exponenți, optând pentru sortarea descendentă. Metoda sortExponent() realizează astfel ordonarea descrescătoare a puterilor monoamelor și este implementată prin instanțierea unei clase SortByExponent, clasă ce implemenează interfața Comparator și conține metoda public int compare(Monom m1, Monom m2), ce returnează diferența puterilor celor două monoame date ca parametri. O altă metodă utilă a clasei Polinom, care a avut menirea reutilizării codului este actualizeazaCoef, ce primește ca parametru un polinom și este o metodă statică ce returnează un nou polinom. Aceasta presupune sortarea monoamelor din care este alcătuit polinomul, după care, cu ajutorul unui while, se verifică puterile a două câte două monoame, iar în cazul în care acestea sunt egale (se întâmplă în urma operațiilor), se actualizează coeficientul, făcându-se suma între cei doi coeficienți inițiali. Noul coeficient este atribuit celui de-al doilea monom din grupare, primul monom ștergându-se din listă, după care ia următoarea pereche(monomul actualizat și cel următor).

În continuare, am aplicat operația de adunare a două polinoame (public static Polinom sumaPoli(Polinom p1, Polinom p2)), care returnează noul polinom rezultat. Metoda a fost construită prin interclasarea ArrayList-urilor de monoame ale polinoamelor date ca parametri, rezultând un vector nou de liste corespunzător polinomului sumă. Apelând metoda de actualizare a coeficientului, va rezulta adunarea polinoamelor. Scăderea a fost realizată pe același principiu, cu observația că înainte de a fi adăugat al doilea vector de liste, fiecare element din acesta a fost înmulțit cu -1 (înmulțirea cu scalar). Astfel s-a obținut diferența între cele două polinoame. Înmulțirea se realizează prin parcurgerea listelor celor două polinoame. Cu ajutorul instrucțiunii while ne poziționăm pe primul ArrayList, după care, folosim instrucțiunea if , ne asigurăm că mai sunt elemente în a doua listă și înmulțim primul element din prima listă, pe rând, cu fiecare componentă din al doilea ArrayList și procedăm la fel cu următorii termeni. De fiecare dată când ajungem la finalul celui de-al doilea șir, ne plasăm pe următorul element din primul și se reia procesul. Metoda de derivare respectă principiul matematic, coeficientul noului monom obținându-se prin produsul dintre coeficientul inițial și puterea corespunzătoare, iar exponentul devenind valoarea celui precedent decrementată cu 1.

Integrarea returnează polinomul obținut prin setarea coeficientului ca fiind raportul dintre vechiul coeficient și exponentul incrementat cu 1 și modificarea puterii prin creșterea valorii acesteia cu o unitate.

Cât despre citirea și parsarea șirului de caractere dat ca input, am folosit tehnica split, care despică, divizează șirul în mai multe subșiruri în funcție de delimitatorul pe care îl impun. Având în vedere faptul că polinomul conține atât termeni pozitivi, cât și negativi, am optat să folosesc funcționalitatea replace, care înlocuiește caracterul ‘-‘ găsit în șir cu ‘+-‘ , pentru a avea posibilitatea de a efectua operații și asupra numerelor negative. Impunând delimitatorul +, am extras fiecare termen din șir, urmând apoi să procesez fiecare monom. Astfel, am mai divizat în substringuri folosind ca delimitator caracterul x, dat fiind că din întreg string-ul noi avem nevoie de coeficient și de putere. În acest fel, primul element din subșir (a[0]) va constitui coeficientul, iar prin plasarea pe caracterul următor și extragerea subșirului parcurs până la acest pas ( a[1].substring(1)) se va obține exponentul monomului. Se creează un monom cu aceste atribute și se adaugă în vectorul-listă al polinomului rezultat. Pentru afișarea polinomului în formatul dorit am apelat la metoda toString().

**public** **static** Polinom readPolinom(String polinom) {

String semn = polinom.replace("-", "+-");

String[] element=semn.split("\\+");//element=monom

ArrayList<Monom> rez=**new** ArrayList<Monom>();

**for**(String s: element) {//extrag monoamele

String[] a=s.split("x");//extrag din monoame elementele

Double coef;

**int** putere;

coef=Double.*parseDouble*(a[0]);//extrag coeficientul

putere=Integer.*parseInt*(a[1].substring(1));//extrag puterea

rez.add(**new** Monom(coef, putere));//adaug monomul in lista

}

Polinom nou=**new** Polinom();

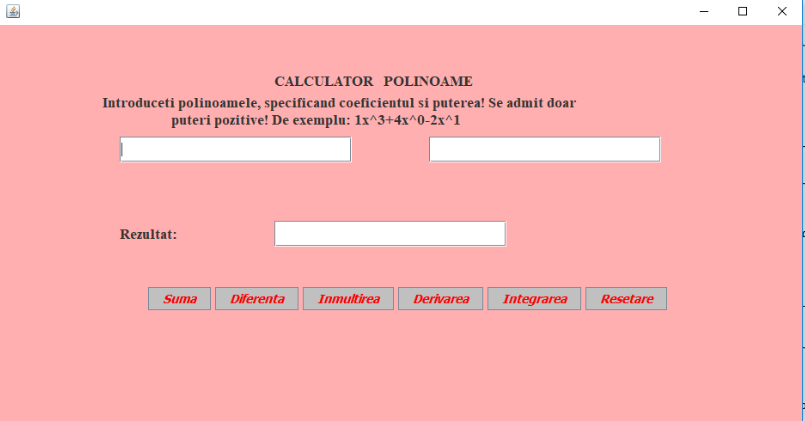
nou.mono=rez;//setez polinomul obtinut

**return** nou;

}

Vederii îi este corespondentă reprezentarea grafică, mai exact, interfața grafică ce interacționează cu utilizatorul final. Rolul este de a evidenția informația obținută înainte ca aceasta să ajungă la controlor. În clasa View am definit și implementat metoda numită meniu(), cu ajutorul căreia am creat fereastra aplicației. Realizarea frame-ului are la bază 3 panouri, unul care conține etichete cu mesaje folositoare utilizatorului, întrucât acestea specifică etapele pocesului de utilizare a calculatorului și de testare a rezultatelor, precum și anumite condiții de folosire sau sugestii care facilitează manipularea acestuia.

De asemenea, primul panel are atașate 3 zone de TextField, primele două destinate introducerii datelor de intrare de cătrre utilizator, iar ultima rezervată rezultatului operației efectuate.



Cel de-al doilea panou este format din mai multe butoane, fiecare cu o funcționalitate specifică. Astfel, am adăugat câte un buton pentru fiecare operație implementată (suma, diferența, înmulțirea, derivarea, integrarea), precum și un buton de resetare cu rolul de ,,curățare” a TextField-urilor în urma comenzii, pregătind terenul pentru ulterioare potențiale operații. Primele 3 butoane sunt folosite pentru efectuarea operațiilor ce necesită 2 polinoame (suma, diferența, produsul), citindu-se din ambele TextField-uri, în timp ce butoanele pentru Derivare și Integrare au ca scop generarea rezultatului operației efecuate doar asupra primului polinom introdus (din primul TextField). Cele două panouri sunt poziționate pe un panel principal (main) care este setat pe fereastră. Această metodă meniu cu ajutorul căreia se realizează interfața grafică este apelată în constructorul clasei View. De asemenea, în această clasă am implementat metoda numită makeButton(String s) care returnează un obiect de tipul JButton în funcție de string-ul primit ca parametru.

**private** JButton makeButton(String s) {

JButton btn1 = **new** JButton(s);

btn1.setBackground(Color.***LIGHT\_GRAY***);

btn1.setForeground(Color.***RED***);

btn1.setFont(**new** Font("Tahoma", Font.***BOLD*** | Font.***ITALIC***, 14));

**return** btn1;

}

Accesul la aplicație este controlat prin Controller, conținutul proiectului luând o formă dinamică, diversificată. Practic, acest controlor realizează conexiunea între Model și View, între logica ce se ascunde în spatele programului și forma vizibilă, concretă a acestuia. În acest fel, în controlor am adăugat ascultători (addActionListener) . Un ascultător este apelat atunci când user-ul interacționează cu interfața, ceea ce provoacă un eveniment. La click pe buton, se invocă metoda actionPerformed() definită în clasa obiectului ascultător. Metodei ise transmite ca parametru un obiect ActionEvent. În această situație, ascultătorii sunt definiți ca și clase imbricate anonime (btn.addActionListener(new ActionListener)).

v.btn.addActionListener(**new** ActionListener(){

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

Polinom p1=**new** Polinom();

**try** {

p1=Polinom.*readPolinom*(v.textField.getText());

}

**catch**(Exception e1) {

JOptionPane.*showMessageDialog*(**null**, "Format incorect!");

}

Polinom p2=**new** Polinom();

**try** {

p2=Polinom.*readPolinom*(v.textField\_1.getText());

}

**catch**(Exception e1) {

JOptionPane.*showMessageDialog*(**null**, "Format incorect!");

}

Polinom p3=**new** Polinom();

v.textField\_2.setText(Polinom.*sumaPoli*(p1, p2).toString());

}

});

1. **Rezultate și concluzii**

În urma acestei teme am reușit să învăț cum să proiectez o interfață grafică pentru un proiect Java, mi-am dezvoltat abilitățile de procesare a string-urilor și am avut oprtunitatea de a-mi pune în aplicare cunoștințele și experiența din domeniul matematicii. Câteva posibile dezvoltări ulterioare ale proiectului meu ar putea fi tratarea cazurilor speciale, precum și realizarea unui design mai evoluat.

1. **Bibliografie**

<https://www.javatpoint.com/java-swing>

<http://users.utcluj.ro/~igiosan/teaching_poo.html>

<https://www.draw.io/>

<http://zetcode.com/tutorials/javaswingtutorial/>