

***TEMA 1***

***Calculator Polinomial***

***Irimes Cristina***

***Grupa 30224***

Cuprins

1. Obiective
2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Proiectare
4. Implementare
5. Rezultate
6. Concluzii
7. Bibliografie

1.Obiective

Proiectati si implementati un calculator polinomial cu o interfata grafica dedicata prin care utilizatorul poate insera polinoame, poate selecta operatia matematica (adica adunare, scadere, multiplicare, divizare, integrare) care trebuie efectuata si vizualizati rezultatul.

Obiectivul principal: Realizarea unei aplicatii ce implementeaza operatii de baza lucrand cu polinoame. Polinoamele sunt construite din monoame, caracterizate de un coeficient si un exponent(grad). Pentru utilizare, se va folosi interfata grafica.

Obiective secundare:

1. Analiza problemei si identificarea cerintelor

* Alegerea structurilor de date si proiectarea claselor dorite

1. Proiectarea calculatorului polinomial

* Realizarea diagramei UML
* Descrierea algoritmilor folositi in realizarea operatiilor

1. Implementarea calculatorului polinomial

* folosirea Model - View – Controller pentru forma o interfata Grafica
* descrierea fiecarei clase impreuna cu metodele si campurile importante
* descrierea interfetei utilizatorului

1. Testarea calculatorului polinomial

* Folosind Junit Test, se vor proiecta cateva teste pentru demonstrarea functionalitatilor

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

Pentru a implementa corect operatiile, inainte de a face acest lucru trebuie sa intelegem conceptele matematice.

Monom = expresie algebrica ce contine un singur termen, continand exponent(grad) si coefficient.

* Exemplu: 2x^2 => monom cu exponentul (gradul) 2 si coeficientul 2

Polinom= expresie matematica formata din una sau mai multe expresii algebrice, in cazul nostrum din unul sau mai multe monoame.

* Exemplu: 5x^5+4x^4-3x^3+2x^2-x+1 => polinom ce contine monoamele: 5x^5, 4x^4, -3x^3, 2x^2, -x(-1x^1) si 1(1x^0)

Operatii realizate folosind polinoame:

* + Adunare
  + Scadere
  + Inmultire
  + Impartire
  + Derivare
  + Integrare

Fiecare operatie in parte ne returneaza la randul sau un polinom, exceptie facand impartirea care ne va returna o lista de polinoame, primul element reprezentand catul, iar al doilea restul.

Modelarea face referire la divizarea intregului proiect in problem mai mici, rezultand astfel o metoda mai usoara de rezolvare. Primul pas in dezvoltare este de a considera polinomul drept o lista de monoame ce este mai usor de folosit. Reprezentarea monomului este de asemenea importanta, exponentul (gradul) va fi de tip int, in timp ce coeficientul float (in momentul in care impartim 2 polinoame sau integram un polinom vom avea erori la calcul daca ii lasam int).

Scenarii si cazuri de utilizare:

1. Introducerea polinoamelor necesare operatiei dorite de realizat (adunare, scadere, inmultire, impartire – cate 2 polinoame; integrare, derivare – doar primul polinom). Acestea se vor introduce in TextField-urile corespunzatoare fiecarui polinom
2. Selectarea operatiei dorite, printr-o singura apasare pe butonul ce contine numele operatiei.
3. Afisarea rezultatului in urma operatiei sau a mesajelor de eroare. Rezultatul se va afisa in JLabel-ul din dreptul celui numit “Rezultat”, in timp ce la operatia de impartire, restul va fi afisat sub campul atribuit catului (“Rezultat”).

* In momentul in care un utilizator introduce in campul specific Polinomului altceva, iar operatia nu se poate realize, va aparea un mesaj de eroare conform caruia utilizatorul este atentionat ca a introdus cel putin un polinom gresit sau in cazul impartirii cu 0 ca aceasta operatie nu se poate realiza.
* Daca utilizatorul introduce un polinom fara sa tina cont de ordinea gradelor sale, programul il va lua ca valid, ordonandu-l in TextField-ul in care se afiseaza rezultatul.

3. Proiectare

In implementarea acestui proiect, am ales sa folosesc modelul MVC. Aceasta cuprinde toate clasele si anume: model (clasa Monom si clasa Polinom), operations(clasa Operatii), ui(clasa View si clasa Controller).

* Pachetul model:
* Clasa Monom: lucreaza cu monomul caracterizat de 2 campuri: tipul int pentru exponent(grad) si tipul float pentru coeficient
* Clasa Polinom: lucreaza cu polinomul reprezentat sub forma de lista de monoame
* Pachetul operations:
* Clasa Operatii: impleenteaza toate cele 6 operatii folosind unul sau 2 polinoame
* Pachetulu ui:
* Clasa View: imaginea intregului proiect unde utilizatorul foloseste aplicatia utilizand butuoanele si field-urile de adaugat polinoamele
* Clasa Controller: implementeaza Listener-ii pentru fiecare buton in parte si ii apeleaza in constructor-ul sau
* Pachetul Test:
* Clasa Test1: implementeaza toate testele utilizand JUnit

In clasa Polinom, elemental principal pe care se realizeaza operatiile, si anume polinomul, este realizat dintr-o lista de monoame folosind ArrayList. Acest tip de structura reduce memoria folosita, monoamele cu coeficient 0 negasindu-se in lista. Asa cum am prezentat mai sus, monomul este reprezentat de grad si coeficient, acestea fiind de tipul int, respective float.

Pentru a sorta polinoamele in ordine descrescatoare a gradelor, reprezentate prin lista de monoame, se va folosi interfata Comparable prin suprascrierea metodei compareTo, aceasta returnand diferenta dintre gradele (exponentilor) in clasa Monom.

1. Implementare

In continuare se vor descrie amanuntit clasele insotite de cele mai importante metode.

* Clasa Monom

Implementeaza conceptele pentru monom, si anume prezinta atributele grad (tipul int) si coefficient (tipul float) ce sunt private. Are definit un singur constructor cu parametrii pentru fiecare atribut ce il caracterizeaza. De asemenea, sunt definite publice metodele getCoef, setCoef, getGrad, setGrad pentru a putea accesa atributele in alta clasa. O metoda importanta este cea toString ce ne va ajuta in viitor cand vom dori sa afisam rezutatul, aceasta fiind apelata si in metoda ToString din clasa Polinom. Aceasta converteste Monomul de forma (grad, coefficient) in expresia “coeficient \* x^grad”.

* Clasa Polinom

Implementeaza conceptele pentru un intreg polinom, si anume prezinta un singur atribut si anume lista de monoame realizata cu ajutorul ArrayList ce este private. Aceasta are definite 2 contructori: unul cu un singur parametru ce face referire la lista de monoame si unul, tot cu un singur parametru, insa de aceasta data parametrul este de tip String. Cel de al doilea constructor, caracterizat de String s, ne ajuta sa transformam sirul de caractere introdus initial in TextField intr-o lista de monoame (intr-un polinom). Acest lucru l-am putut realiza cu ajutorul unui pattern (Regex) ce imparte stringul in grupuri de monoame, tot de tipul String, urmad ulterior sa fie dezvolate si transformate in Monoame sub forma (grad, coefficient).

Pattern-ul folosit este de forma : "([+-][0-9]\*)([xX]\*([\\^][0-9]+)\*)", unde:

* ([+-][0-9]\*) permite folosirea a cate cifre dorim, avand la inceput + sau –
* ([xX]\*([\\^][0-9]+) permite folosirea variabilei X sau x, urmata de al treilea grup ce reprezinta puterea/gradul lui x

Inainte de a extrage din String polinomul, verificam daca sirul de caractere de la intrare incepe direct cu cifra/numar sau cu semn. In cazul in care incepe direct cu cifra/numar, vom concatena sirul cu semnul “+” pentru a fi recunoscut ulterior de pattern drept un nou monom.

De asemenea, sunt definite publice metodele getM, setM pentru a putea accesa lista de monoame in alta clasa. O metoda importanta este cea toString ce apeleaza la randul sau metoda toString din clasa monom si ne ajuta sa transformam polinomul intr-un sir de caractere pentru a il putea afisa. La sfarsitul acestei metode, polinomul rezultat este ordonat descrescator prin utilizarea metodei sort din interfata Comparable.

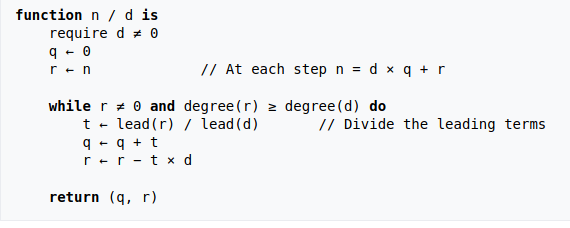
* Clasa Operatii

Aceasta clasa are implementate toate operatiile pe polinoame prin metode diferite: adunare, scadere, inmultire, impartire, derivare si integrare.

Adunarea – prezinta 2 parametrii de tipul Polinom si returneaza un nou Polinom. Initial, parcurgem cele 2 liste de polinoame a fiecarui polinom. La o prima traversare, vom aduna coeficientii termenilor ce au acelasi grad, memeorand in variabila k daca a fost adaugat sau nu in noua lista de Monoame, in cazul in care k este 0, inseamna ca gradul primului polinom nu se gaseste printer gradele celui de al doilea polinom, termenul ce il contine fiind adaugat in lista de monoame. In cea de-a doua traversare, punem pe primul plan al doilea polinom si verificam dupa aceeasi metoda, ce coeficienti nu se gasesc in primul polinom, iar cele ce nu se gasesc se adauga in lista de monoame.

Scaderea – este implementata in acelasi mod precum adunarea, prezentand 2 parametrii de tipul Polinom si returnand un nou Polinom. Singura diferenta intre adunare si scadere este faptul ca la prima traversare coeficientii se scad, iar la a doua traversare, monoamele ce nu au fost gasite in primul polinom si se gasesc in cel de-al doilea polinom, vor fi adaugate in lista de monoame avand coeficientul negat.

Inmultirea – asemenea operatiilor de mai sus, prezinta 2 parametri de tipul Polinom si returneaza un nou Polinom. Aceasta se realizeaza printr-o singura parcurgere a celor 2 polinoame in acelasi timp, coeficientii fiind inmultiti iar gradele adunate. Aici putem avea o repetare a unor monoame ce au acelasi grad, dar coeficienti diferiti. De aceea cautam in noua lista daca se gaseste gradul pe care dorim s ail introducem. Daca se gaseste, vom aduna vechiul coeficient al monomului cu gradul nostrum si coeficientul noului termen si apoi il adaugam in lista noua de monoame.

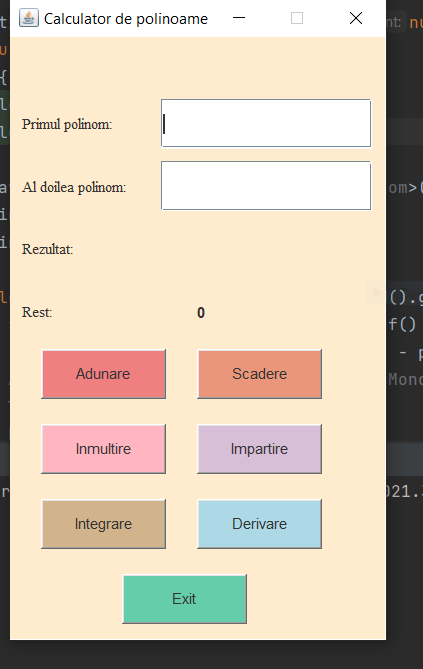
Impartirea – are si aceasta 2 parametrii de tipul Polinom, insa fata de celelalte metode, aceasta este singura care returneaza o lista de Polinoame, primul element fiind reprezentat de cat, iar al doilea de restul impartirii. Am implementat algoritmul “polynomial long division”, adaptand pseudocodul la limbajul Java.

Derivare – prezinta un singur parametru de tipul Polinom, returnand tot un Polinom. Printr-o singura traversare a polinomului, vom adauga in noua lista de monoame fiecare monom din polinomul dat ca parametru la care schimbam gradul si coeficientul. Gradul va deveni grad-1, iar coeficientul va fi coeficient\*grad.

Integrare – se realizeaza dupa acelasi mod ca metoda anterioara (Derivare), diferenta consta in calculareea gradului si coeficientului. Gradul va deveni fostul grad + 1, iar coeficientul va deveni coefficient/(coefficient+1).

* Clasa View

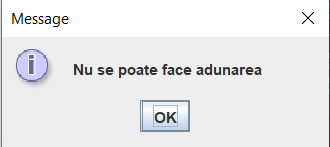
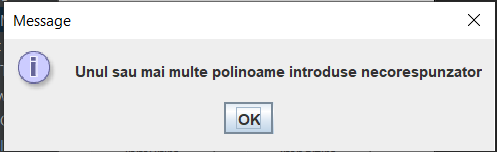
Aceasta extinde JFrame. Aici sunt definite ca atribute private fiecare buton de care avem nevoie, fereastra propriu-zisa, textField-ul unde va fi scris rezultatul (catul in cazul impartirii), respective restul si cele 2 textField-uri unde introducem polinoamele asupra carora dorim sa realizam operatiile. Dupa ce setam dimensiunea, locul si numele fiecarui element, interfata va arata astfel:

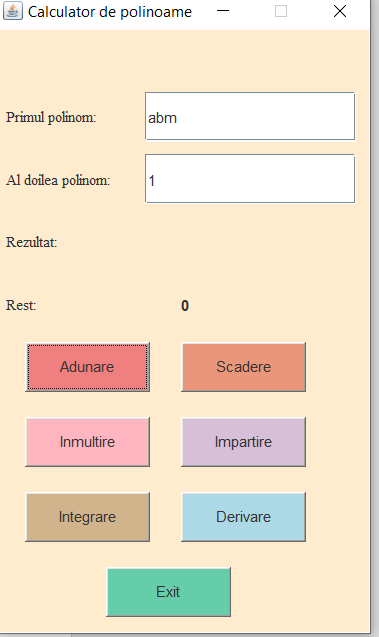


Aceasta cuprinde si metodele getPol1(), getPol2(), setRez(String s) si setRest(String s) cu ajutorul carora preluam din textField-uri polinoamele, respective inseram rezultatul si restul (cand e cazul) ce trebuie afisat. Mai mult, exista metoda close() ce ne ajuta in inchiderea ferestrei apasand pe butonul Exit.

Pentru fiecare buton din interfata este nevoie de ActionListener pentru a putea accesa butoanele si pentru a ne fi efectuate operatiile.

Exemple introducere date necorespunzatoare:





* Clasa Controller

Are drept atribute Interfata I si Operatiile op pentru a putea lega butoanele de ceea ce ar trebui sa faca. In constructorul acestei clase, sunt incluse si ActionListenerele pentru toate butoanele. In aceasta clasa gasim metoda verificare prin care vedem daca sirul de caractere trimis drept parametru daca este de forma unui polinom sau nu, iar in cazul in care sirul de caractere nu va corespunde cu forma unui polinom, se va afisa un mesaj de eroare conform caruia suntem atentionanti ca “unul sau mai multe polinoame sunt introduce necorespunzator” si ca “operatia nu se poate realiza”. De asemenea, este tratat si cazul in care la impartire al doilea polinom este 0.

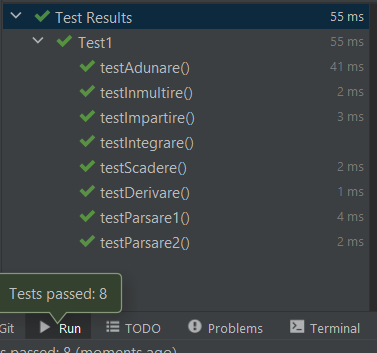
* Clasa Application

Conform structurii MVC, acest proiect a fost dezvoltat in jurul ei. Astfel, fiecare clasa se afla in pachetul sau corespunzator. Aceasta clasa instantiaza interfata, urmata de instantierea controllerului ceea ce face trimitere indirect la restul de clase si operatii ce le-am folosit in dezolvatrea proiectului.

1. Rezultate

Pentru partea de testare, am ales sa folosesc JUnit5 si clasa Test1. Fiecare metoda in parte reprezinta testarea unei anumite operatii, cat si testarea parsarii String-ului primit drept intrare in polinom.

Fiecare metoda de test este precedata de adnotarea @Test. Cele 2 polinoame instantiate la inceputul clasei vor fi folosite in toate metodele de testare. Rezultatul operatii a carui test este realizat va fi comparat prin metoda assertEquals cu string-ul ce reprezinta rezultatul asteptat. Daca cele doua mentionate anterioare sunt egale, testul va avea succes, altfel va afisa fail.



1. Concluzii

Aplicatia respecta toate cerintele prezentate in tema. Realizarea ei m-a ajutat sa inteleg mai bine notiunile prezentate atat la cursul de Programare Orientata Obiect, cat si cele prezentate pana in momentul de fata la Tehnici de Programare.

Am analizat in special structura MVC si realizarea unui Gui in Java. In plus, am invatat cum sa folosesc Regex pentru a valida ceea ce introduce si pentru a converti in String. Am folosit JUnit pentru a realiza teste pentru a vedea imediat daca o metoda are rezultatul asteptat sau nu.

Desigur, acest calculator s-ar putea dezvolta ulterior, realizand si alte operatii, nu doar cele ce au ca argument polinoame.

1. Bibliografie

***Regex:***

<https://regex101.com/>

<https://stackoverflow.com/questions/36490757/regex-for-polynomial-expression>

***Junit:***

<https://www.jetbrains.com/help/idea/create-tests.html>

***Diagram UML:***

<https://app.diagrams.net/>