CALCULATOR DE POLINOAME

-TEHNICI DE PROGRAMARE TEMA 1 -

Diaconu Andrei
Grupa 30227

CUPRINS

1 Introducere

Polinoamele sunt printre cele mai cunoscute elemente matematice regasite totodata si la nivel de programare . Opeartiile nu ridica un grad inalt de dificultate . Prin intermediul acestora putem combate probleme mult mai complexe , regasite in alte contexte.

Intru acest scop am decis ***obiectivul lucrarii*** care cuprinde crearea unei interfete grafice (*vezi package Interface -> View 7)* care permite introducerea a doua polinoame sub forma de input-text , iar in functie de decizia utilizatorului prin butoane controlate (*vezi package Interface -> Controller 9)* se pot efectua calcule asupra lor , precum : *adunare* - , *scadere* - , *inmultire* - , *impartire* - , *integrare* - , *derivare* - . Utlimele 2 operatii se refera strict la un singur polinom (mai exact la primul input) *(vezi package Logical -> Model -> 12)* . Inputurile sunt introduse sub forma de text (String) asadar este necesara parsarea acestora sub forma de monoame (*vezi package Logical -> Monom 11) ,* iar monoamele retinute intr-un ArrayList intr-o clasa specifica *(vezi package Logical ->Polinom 10).*

2 Analiza problemei

Cand ma gandesc la analiza problemei , ma refer la ideea principala , schitata in mare , care functioneaza pe baza unor input-uri abstracte. Prin intermediul lor gasim comporamentul problemei si cum ar trebui sa decurga rezolvarea.

Metoda conceptelor sau bottom-up design desi prezinta oarecum o complexitate ridicata eu o consider benefica deoarce gasesti relativ usor componentele / structurile de care ai nevoie pe parcursul rezolvarii. De asemenea folosesc ideea de subiect-predicat , care consta in definirea numelor de clasa (subiect) si actiunile ce pot fi implementate de subiect (predicatele) reprezinta de fapt methodele din interiorul acesteia.

Odata regasit firul principal al aplicatiei trebuie sa avem in vedere ca aplicatia ce o implementam va fi accesibila publicului larg de utilizatori (unii pot fi mai obisnuiti cu tehnologia decat altii). Asadar design-ul interfetei trebuie sa fie unul concis , comunicarea cu aplicatia sa fie una cat mai simplista.

Pe baza acestui principiu am decis ca input – urile sa fie de fapt niste simple texte , fara a-l determina pe utilizator sa foloseasca niste functii specifice pentru ridicarea la putere sau gestionarea coeficientilor . In acest scop am si adus in partea dreapta a interfetei o tastatura ce cuprinde caracterele permise sa fie introduse , sa nu intalnim cazuri in care acesta introuduce aleator alte litere care automat vor fi nerecunoscute de utilizator conducand spre o eroare de compilare.

Daca atat ideea principala este clar elaborata si interfata reprezinta un GUI usor de inteles atunci trebuie sa ne ganndim la actiunile utilizatorilor in momentul rularii aplicatiei, asadar aducem in cunostiinta de cauza *cazurile de utilizare.*

- **CAZURI DE UTILIZARE**

Aceast punct trebuie vazut din perspectiva unui utilizator , eu ca programator stiind exact cum va decurge mersul problemei. Asadar un utilizator poate introduce:

- Doua polinoame , in cele 2 input-uri , in ordine descrescatoare a exponentilor fara ca acestia sa se repete . In acest moment ele sunt perfect introduse si orice operatie ar alege s-ar executa cu succes si afisa rezuultatul in JTextFiled-ul corespunzator raspunsului. Daca totusi acesta alege o operatie compatibila doar pe un polinom (Integrare / Derivare) nu se va genera vreo eroare ci se va executa operatia, intotdeuna, asupra primului input.
- Doua polinoame , ordonate descrescator si fara sa se repete vreun exponent, iar unul din ele sau chiar ambele , contin mai multe simboluri reprezentatic puterii (^) consecutive , din nou nu se va genera eroare , chiar daca este oarecum gresit , regex-ul din monom face astfel incat primul caracter dupa acea serie de ” ^ ” sa reprezinte exponentul , iar operatiile se pot efectua in continuare in mod aferent.

- Doua polinoame , in ordine aleatorie a exponentilor cu dubluri , tripuluri , etc. Acestea in momentul memorarii vor fi actionate niste functii care determina eficientizarea acestor regasindu-se fiecare exponent o singura data. Iar la momentul afisarii acestora se aplica si functia de ordonare implementat pe baza Collections.
- Un singur polinom si solicita adunarea , scadera sau oricare functie care necesita 2 polinoame. Operatia se va executa iar ca al doilea operand se va lua polinomul 0.

- **MODELAREA**

Ajungand la modelare , aceasta structura se refera la parsarea si la sectiunea ce descrie transformarea unor simple inputuri in ArrayList-uri si operatii pe baza acestora.

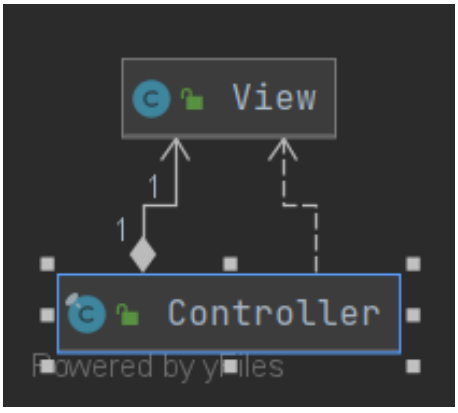
Clasa Monom care primeste text-ul din polinom il separa pe acesta in monoame , retinand la al catelea monom se afla . Parsarea coeficientilor se face cu tot cu semn , in coeficient este retinut si semnul +/-.

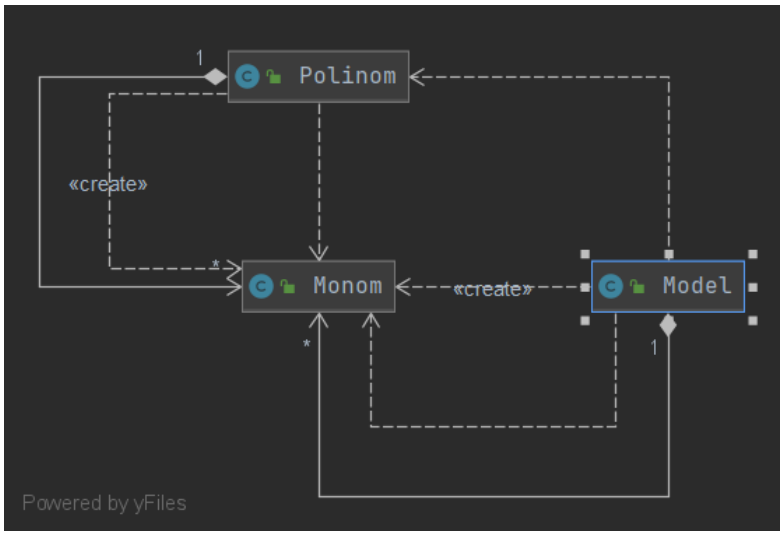
3. Proiectare si Implementare

Proiectul contine 5 clase de baza structurate in 2 pachete . Pe langa acestea se afla si clasa Main care actioneaza merge-ul lor.

Am folosit tipuri de date cunoscute si tipuri de date precum ArrayList , atat in clasa Polinom care gestioneaza Monoamele cat si in Clasa a carei metode prelucreaza o astfel de Colectie. Listele sunt parcurse cu for-each.

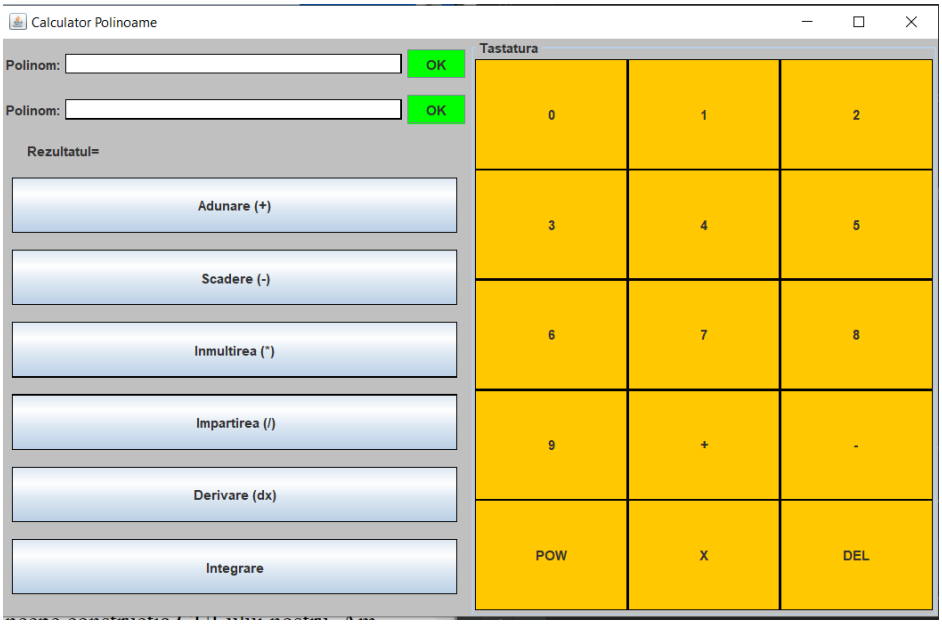
S-a respectat de asemenea structura de MVC (model view controller) deoarcere in view se creeaza grafica proiectului , in model se modeleaza ideile de lucru iar in controller sunt listenerii fiecarui buton din interfata generata in clasa View.





- ***Pachetul Interface***

-Clasa View



In interiorul acestei clase se intampla toata grafica proiectului folosing JFrame dar si Javax.swing . Clasa incepe cu declaratiile necesare , iar din constructor se apeleaza functia dooLayout() care de fapt incepe constructia GUI-ului nostru. Am folosit o structura de GridLayout cu o dimenisune de 950x600.

Frame-ul la randul lui l-am structurat in doua Panel-uri : Unul detine strctura de inputuri , output , si opeartii posibile , iar cea de-a doua o tastatura virtuala. Pentru o eficienta cat mai buna , am decis ca tastatura sa fie reprezentata de un arraylist de butoane care poti fi

accesate usor . Numele fiecarui buton reprezinta de fapt valoarea lui in sine dupa ActionListener , mai putin “del” si “^” care au avut nevoie de o implementare separata.

Privind partea stanga observam ca toate componentele sunt sub forma flowlayout . In dreptul celor 2 input-uri se afla butoanele de OK , care reprezinta de fapt pornirea procesului in spatele interfetei. Transmite textul procesat catre Modelare. Toataodata am folosit metoda de focusListener care ne determina cand focus-ul a fost primit sau pierdut. Acest lucru ne ajuta sa stim unde se vor scrie valorile butoanelor de pe tastatura virtuala. Controlul lor este setat un boolean isSelected. P1,P2 reprezinta JTextFiled-uri cunoscute de utilizator ca si input-uri.

```
p1.addFocusListener(new FocusListener() {
    @Override
    public void focusGained(FocusEvent e) { isSelected = true; }

    @Override
    public void focusLost(FocusEvent e) {
    }
});

p2.addFocusListener(new FocusListener() {
    @Override
    public void focusGained(FocusEvent e) { isSelected = false; }

    @Override
    public void focusLost(FocusEvent e) {
    }
});
```

Metodele care contin numele Layout se refera la aranjarea lor in GUI , insa cand este vorba de utilizarea lor (functia operatii) intervine in joc Controllerul.

- Clasa Controller

Clasa aceasta este una importanta deoarce de aici pleaca comenziile utilizatorilor . Este o clasa ce contine 2 variabile instanta statice , care reprezinta de fapt Model si View-ul proiectului nostru.

Sunt delcarate statice deoarece in interiorul ei am construit Inner Classes care fiecare la randul ei reprezinta un Button din View. Acestea implementeaza ActionListener si au o methoda care trebuie suprascrisa si reprezinta actiunea deja savarsita (ce ar trebui sa se intample dupa ce utilizatorul a apasat butonul).

Referitor la cele 2 Ok-uri de validare am folosit un pop up care ne specifica exact cum arata polinom de abia introdus . Avand in vedere scenariul ca utilizatorul poate executa mai multe operatii una dupa alta trebuie sa avem in minte ideea de resetarea a output fieldului , de aceea la fiecare validare de text nou , in output nu mai avem nimic.

Restul claselor din aceasta clasa implementeaza acelasi lucru , mai exact foloseste Modelul pentru a face operatia respectiva si apeleaza functia de printare. La integrare exista totusi o mica diferenta deoarce proiectul in sine se refera la operatii cu polinoame pe numere intregi asadar cand vine vorba de integrare si avem fractii , acestea le gestionez in modul text . As fi putut lucra cu zecimale si cast la float insa consider ca pentru orice utlizator este mult mai la indemana lucrul cu fractii fiind mult mai usor visibile si pe mai departe de lucrat cu ele.

- ***Pachetul Logical***

-Clasa Polinom

Se bazeaza in mare parte pe ArrayList<Monoame>

Private final String text este text-ul primit de la utilizator si in metoda createPolinom se trimite acest text catre Clasa Monom de atatea ori cate monoame sunt . Numarul monoamelor este determinat de o alta metoda tot cu nume specific getNumarMonaome() care foloseste Regex. Clasa Monom este totusi accesata cu un index ca sa stie la ce parte din intreg textul s-a ajuns (la ce monom).

Tot aici se discuta si unul din cazurile de utilizare speicificate mai sus. Methoda eficientizare strabate acelasi polinom de doua ori. Retinand fiecare element din prima parcurgere si comparandu-l cu toate restul din cea de a doua parcurgere (care au acelasi exponent). Notam intr-un array obisnuit de int-uri valorile exponentilor vizitati, deoarce lucram pe acelasi polinom- acelasi arraylist si trebuie sa evitam cazul in care eficentizam 2 monaome cu acelasi exponent de 2 ori. Asadar daca un monom a fost deja vizitat si schimbat coeficientul in functie de exponent , aceluia putem sa ii atribuim coeficientul 0 , iar mai apoi cand este vorba de prelucrarea monoamelor , mereu vom exlcude cazurile cand exista un coeficient 0 .

```java
public void eficientizare(ArrayList<Monom> monoame) {

    int[] visitedExp = new int[100];
    boolean isVisited = false;
    int index=0;

    for(Monom m : monoame)
        for (Monom m1 : monoame)
            if (m1.getExp() == m.getExp() && m != m1) {
                for (int i = 0; i < index; i++)
                    if (visitedExp[i] == m1.getExp()) {
                        isVisited = true;
                        break;
                    }
                if (!isVisited) {
                    m.setCoef(m.getCoef() + m1.getCoef());
                    visitedExp[index++] = m.getExp();
                } else{
                    isVisited = false;
                    m.setCoef(0);
                }
            }
}
```

- Clasa Monom

Cea mai complexa clasa si cea mai imporatanta deoarece aici se face parsarea

Pe langa gettere si settere mai intalnim methoda createMonom() care este apelata cu un index , exact ceea ce am vorbit la punctul anterior. Ca pattern compile am folosit “([+-]?[^-+]+)” , iar ca matcher text-ul in sine. Pentru efcientizare la fiecare matcher.find() daca nu ne aflam la monom-ul concret de care aveam nevoie (stabilit de parametrul index) treceam mai departe . In cazul afirmativ am folosit 2 StringBuilderi care retineau exponentul respectiv coeficientul pe baza caracterului X. In cazul in care caracterul x lipsea aveam de a face cu termenul liber si retineam doar coeficientul iar exp era setat la 0.Daca dupa x urma un semn +/- aveam de a face cu un monom de grad 1 asadar setam exponentul aferent. Toate aceste cazuri le tratam la sfarsit prin lungimea StringBuilderului si un boolean.

Tot aici aveam cazul in care utilizatorul scria simplu “ -x ”, asadar coeficientul conform algoritmului implementat mai sus reiesea un caracter , asadar am luat cazul in paralel si setam –1 ca si coeficient.

La final , setExp si setCoef am fost apelate cu ajutorul lui Integer.parseInt() iar StringBuilder transformat in String prin toString().

- Clasa Model

Modeleaza proiectul in sine deoarce aici se afla rezolvarile actiunilor solicitate de Controller primite din interfata View.

Am implementat pe rand cate o metoda pentru fiecare operatie. Fiecare metoda in sine prelucreaza un arraylist declarat la inceput de clasa.

Pentru operatia de adunare nu este nimic decat parcurgerea a doua for-uri, for-eachuri si in momentul in care 2 exponenti sunt egali le adunam coeficientii . Daca un monom nu isi gaseste in celalalt polinom perechea isi va pastra forma initiala. Totusi trebuie sa avem grija deoarce exista monoame care nu au fost luate in calcul. Cele din al doilea polinom care nu si au corespondent in primul , pentru aceasta fiind aceeasi metoda ca si la Scadere am facut o metoda separata numita checkSkipMonom care verifica sa nu lipseasca nimic din adunare sau scadere. Scaderea are totusi ceva in plus deoarece la parcurgere , incepand cu polinomul 2 (cu scazatorul) trebuie sa le dam coeficientilor fiecarui monom un minu in fata , in rest operatiile decurg la fel.

La inmultirea avem nevoie de adunare deoarcere aceasta operatie consta intr-o serie de adunari. La derivare se inmultesc pur si simplu coeficientul cu exponentul si se scade cu o unitate exponentul.

In ceea ce priveste Integrala noi incepand cu numere intregi acest proiect , am decis sa lucrez in mod String(text) deoarce am folosit frarctiile in locul zecimalelor care necesitau cast la float , un lucru mai greu de verificat de utilizator . Am luat aceasta decizie pe baza ideei ca majoritatea utilizatorilor prefera fractiile in loc de zecimale.

Afisarea ArrayList-ului am facut-o in toString surpascriind aceasta methoda , ea fiind accesata in Controller prin intermediul unui obiect model.

4.Concluzii

Ajungand la finalul acestei teme am realizat ca am pus in practica, cap la cap, multe informatii dobandite creeand un proiect complex. Totusi negestionarea timpului corect nu mi-a permis implementarea JUnit-ului.

Indicatiile au fost folositoare deoarce nu pornisem pe ideea de clasa separata Monom, ci a unui HashMap <Key,Value> care genera o complexitate foarte mare in cazul in care un utilizator introducea exponentul 100, sau o valoare mare.

5 Bibliografie

- https://users.utcluj.ro/~igiosan/teaching_poo.html - 10.03.2021 16:35
- https://stackoverflow.com/questions/36490757/regex-for-polynomial-expression - 11.03.2021 17:00
- https://ro.wikipedia.org/wiki/Model-view-controller - 14.03.2021 19:30
- https://www.smartdraw.com/uml-diagram/ - 16.03.2021 17:30