

Calculator de polinoame

Documentatie

Andrei Burian

Grupa 30229

Cuprins

1. Obiectiv ............................................3
2. Studiul problemei .............................3
3. Proiectare .........................................6
4. Implementare...................................7
5. Rezultate ........................................10
6. Concluzii .........................................10
7. Bibliografie ....................................10
8. Obiectiv

Obiectivul acestei lucrari de laborator este de a proiecta, implementa si testa o aplicatie cu interfata grafica (UI) care poate realiza calcule cu polinoame. Practic, acest proiect reprezinta un calculator de polinoame. Polinoamele sunt in functie de o singura variabila, iar calculele realizate sunt: adunarea, scaderea, inmultirea, impartirea, derivata si integrala. Ca orice sistem de calcul, aplicatia noastra are date de intrare si date de iesire, care sunt procesate si afisate dupa efectuarea calculelor.

Interfata grafica are rolul de a face interactiunea dintre utilizator si aplicatie, cat mai simpla si intuitiva.

* 1. Obiective secundare

Primul obiectiv este acela de a crea clasa Monom, iar pe baza acesteia sa creez clasa Polinom, fiecare polinom avand o lista de monoame in care se memoreaza polinomul propriu-zis.

Cel de-al doilea obiectiv este realizarea operatiilor pe polinoame, implementarea acestora facandu-se cu metode in clasa Polinom.

Al treilea si ultimul obiectiv este realizarea unei interfete grafice (UI), in care se pot introduce datele de intrare si unde se si afiseaza datele de iesire, toate acestea realizandu-se pe o arhitectura de tipul MVC (model – view – controller).

1. Studiul problemei

2.1 Analiza problemei

Cea mai ampla problema a fost implementarea propriu-zisa, faza de proiectare si modelare a aplicatiei fiind necostisitoare din punctul de vedere al timpului. Cele mai ample (sau dificile) probleme la implementare au fost realizarea operatiilor pe polinoame si crearea interfetei grafice, care trebuia conceputa pentru orice tip de utilizator si care ia in calcul toate scenariile posibile de utilizare.

2.2 Modelare

Clasa Monom a fost construita in asa fel incat pe polinoamele din care fac parte sa se poata efectua operatii de impartire si derivare, care denota necesitatea coeficientilor reali, astfel coeficientul din clasa Monom este de tip float. Cu toate acestea, daca rezultatul nu are coeficienti reali acesta va fi afisat cu coeficienti intregi, iar caz contrar coeficientii vor fi aproximati la 2 zecimale.

Clasa Polinom consta, de fapt, dintr-un singur atribut, si anume, un ArrayList de obiecte de tip Monom. De altfel, aici sunt definite toate metodele care calculeaza fiecare operatie pe polinoame, unele dintre acestea avand modificatorul de acces „private” deoarece sunt metode folosite doar in acesta clasa si nu e nevoie sa fie vazute din exterior. De asemenea, toate campurile claselor sunt de tipul „private” respectand paradigma programarii orientata pe obiecte, acestea putand fi accesate sau modificate doar prin intermediul getter-ilor si setter-ilor.

2.3 Use case

Aplicatia consta intr-o fereastra care se deschide, acesta creand un pod de legatura intre un utilizator oarecare si sistemul de calcul din spate.

Se specifica la deschidere primul pas pe care utilizatorul trebuie sa il faca: selectarea unei operatii. Dupa acesta operatie interfata grafica se schimba in functie de nevoi, de exemplu la operatiile cu un singur operand ramane un singur camp pentru input, iar la impartire apare un al doilea camp de output, pentru rest.

Polinoamele introduse trebuie sa se supuna urmatoarelor reguli de scriere:

* Niciun polinom nu poate fi nul
* Fiecare monom trebuie sa aiba o putere (chiar daca e 1), inafara de monomul de gradul 0
* Puterea este precedata intotdeauna de notatia „^”
* Polinoamele vor fi intotdeauna in functie de variabila „x” (x mic)
* Gradele monoamelor si coeficientii pot fi doar intregi

Cu toate aceste restrictii, utilizatorul are parte de flexibilitate din partea aplicatiei cand vine vorba de input:

* Polinomul nu trebuie sa aiba gradele in ordine descrescatoare sau crescatoare
* Elemente cu acelasi grad se pot repeta, chiar daca se simplifica

Interfata grafica dupa ce este pornita aplicatia:

Graphical user interface, application

Description automatically generated

In acest moment utilizatorul trebuie sa aleaga operatia pe care doreste sa o efectueze.

UI-ul poate lua mai multe forme, in functie de ce doreste utilizatorul sa calculeze, evidentiind operatia selectata:

Graphical user interface, application, table

Description automatically generated

Table

Description automatically generated

Graphical user interface

Description automatically generated

2.4 Cazuri de utilizare

Aceasta aplicatie a fost proiectata in asa fel incat sa fie un singur scenariu de utilizare:

1. User-ul selecteaza operatia dorita
2. Se introduc operanzii (1 sau 2, in functie de operatia selectata la pasul 1)
3. Se apasa pe butonul „Calculate!” iar rezultatul/ele va/vor aparea in dreptul etichetelor pentru rezultat/e
4. Proiectare

Proiectul este bazat pe arhitectura MVC (Model – View – Controller) care este o tehnica specifica programarii orientate pe obiect. Avem, de asemenea, si un pachet, care cuprinde clasele Polinom si Monom, pentru a putea fi integrate mai usor in contextul arhitecturii MVC.

Clasa View contine interfata grafica si toate elementele ei. Aceasta clasa nu are acces la celelalte clase, doar viceversa. M-am folosit aici de biblioteca swing si biblioteca awt pentru a putea modela elementele interfetei grafice si pentru a le putea alinia si aranja pentru un aspect cat mai placut.

Clasa Model primeste, ca parametru, in constructor un element de clasa View, pentru a putea interactiona cu interfata. Aceasta clasa are metode pentru fiecare operatie, dar are inclusa si clasa Polinom din pachetul Polynomial. Aceste metode au ca date de intrare string-uri, care sunt transformate in obiecte din clasa Polinom si pe care se efectueaza operatia corespunzatoare. Rezultatul operatiei este convertit in text si afisat pe interfata grafica prin intermediul etichetelor rezultat si clasei View, primita ca parametru la instantiere.

In final, avem clasa Controller, care contine ActionListener-ii pentru fiecare buton de pe interfata. Mai intai sunt creati si pe urma implementati. La implementare interfata este modificata in functie de butonul de operatie apasat, iar cand se apasa butonul de calcul se obtin stringurile scrise de utilizator si se transmit la model pentru efectuarea operatiilor si afisarea lor pe interfata.

Mai jos este diagrama UML a proiectului:

Text

Description automatically generated with medium confidence

Se poate observa faptul ca in clasa Controller am creat mai multe clase, una pentru fiecare ActionListener pentru a-i putea implementa separat in parte, in clasa Controller si nu in clasa View, unde este descrisa interfata grafica cu butoanele respective.

1. Implementare

4.1 Monom

Campurile clasei Monom sunt coeficient si putere. Coeficient e de tip float, iar putere de tip int. In aceasta clasa sunt doar getter-i si setter-i, dar si un Override la metoda compareTo din interfata Comparable. Clasa Monom implementeaza interfata Comparable deoarece se doreste sortarea monoamelor dupa putere (grad) mai departe, cand trebuie manipulate polinoamele.

4.2 Poliom

Singurul camp al clasei Polinom este un ArrayList care memoreaza elemente de tip Monom. Avem si un getter si un setter pentru aceasta lista. Mai departe sunt metodele de prelucrare a polinomului, incepand cu crearePolinom, care dintr-un string creaza un Polinom.

La inceputul metodei crearePolinom inlocuiesc fiecare „–” (minus) din string-ul primit ca parametru cu „+-” pentru a putea sectiona sirul in functie de un singur separator: +, ceea ce se si intampla in continuare. Rezultatul acestei operatii este un vector de string-uri, fiecare reprezentand un monom. Parcurg pe urma fiecare string (monom) si il sectionez din nou, doar ca de aceasta data folosesc ca separator „x^”, pentru a putea separa coeficientul de putere. Dupa aceea, cu rezultatele obtinute creez un nou element de tip Monom, pe care il adaug la lista listaMonoame, care este, practic, polinomul. Dupa ce se termina toti acesti pasi pentru fiecare monom in parte simplific polinomul si il normalizez.

In metoda de simplificare, la inceput, inainte de toate, sortez lista de monoame, descrescator dupa putere. Dupa aceea parcurgem lista manual (toate parcurgerile inafara de acesta sunt realizate cu for each, aici avand nevoie de mai multa flexibilitate asupra iteratorului), iar in cazul in care puterea elementului urmator este egala cu cea a elementului curent, se parcurge lista pana se ajunge la final sau pana puterea este alta si se aduna toate elementele cu aceasta putere in primul element cu puterea respectiva, iar monoamelor parcurse (inafara de primul) le sunt setate coeficientul pe 0.

La normalizare se sterg elementele care au coeficientul 0.

Pe urma, avem metoda toString care converteste polinomul intr-un String. Se parcurge lista de monoame si se adauga, pe rand, fiecare element (coeficient + „x^” + putere). In cazul in care coeficientul e pozitiv, i se adauga un „+” in fata, sau daca este intreg se afiseaza ca intreg, iar daca este real se rotunjeste la 2 zecimale. De asemenea, daca coeficientul este 1 nu se insereaza nimic in sirul rezultat, iar daca este -1 se insereaza doar un minus. Tot aici avem o verificare pentru cazul in care se face operatia de integrare. Daca se procedeaza astfel, multimea constantelor am marcat-o prin valori aparte, si anume, coeficientul este valoarea minima pe care o poate lua float-ul, in pereche cu puterea care este numarul cel mai mic care se poate memora pe int. Daca acesta pereche este intalnita atunci se va adauga la string-ul rezultat doar „+C”.

In metodele pentru operatii am folosit urmatorul rationament legat de polinoamele rezultat: le-am alocat memorie noua, chiar daca monoame din polinoamele operanzi se regasesc si in rezultat, pentru ca atunci cand se fac mai multe operatii una dupa alta sa nu interfereze faptul ca o referinta data la un polinomul rezultat de la monom din polinomul operand este modificata, reflectandu-se aceasta modificare in polinomul operand.

La adunare am adaugat monoamele din listele celor doi operanzi in aceeasi lista rezultat dupa care am simplificat si normalizat rezultatul. Toate operatiile inafara de impartire, derivata si integrala memoreaza rezultatul in obiectul curent (this).

La scadere am procedat la fel ca la adunare, doar ca la al doilea polinom i-am inversat coeficientii prin intermediul unei functii private de inversare.

Inmultirea a fost implementata cu doua for each-uri, pentru a inmulti fiecare element cu fiecare. Pe urma lista s-a simplificat si normalizat.

La impartire mi-am facut o copie a deimpartitului, pentru ca trebuie modificat. Am lucrat intr-o bucla care se oprea cand gradul deimpartitului devenea mai mic decat cel al impartitorului. Acolo faceam impartirea propriu-zisa, memoram monomul si il adaugam la cat si pe urma inmulteam acel monom cu impartitorul si memoram rezultatul intr-un Polinom auxiliar. Pe urma scadeam acel auxiliar din deimpartit si se reluau operatiile descrise mai sus pana cand conditia de integritate a buclei era incalcata. La final deimpartitul ramas era memorat in Polinomul rest.

4.3 View

Interfata a fost implementata folosind biblioteca swing din Java. Mi-am creat toate elementele interfetei ca variabile ale clasei, pe care le-am si initializat. In constructor, le-am adaugat doar JPanel-urilor si JFrame-ului pentru a rezulta interfata. M-am folosit de mai multe JPanel-uri si un JFrame, in care am adaugat toate JPanel-urile la final. Panel-urile le-am impartit intuitiv: „panelIO” pentru campurile de intrare si iesire, „operationTitle” pentru textul dintre butoane si etichete, „operations” pentru butoanele cu operatii si „calcButton” pentru butonul de calcul. La JFrame i-am setat un minimumSize (oricum dupa fiecare operatie se redimensioneaza in caz ca toate elementele nu mai incap in frame). M-am folosit de layout-ul GridLayout pentru a-mi structura campurile de I/O si butoanele. La final toate panel-urile le-am adaugat intr-un panel principal „pGeneral”, pe care pe urma l-am adaugat in JFrame.

4.4 Model

In model am 5 campuri: un view si 4 Polinoame (2 operanzi si 2 rezultate). Aici am implementat pe clasa Polinom realizarea operatiilor, folosindu-ma de metodele din clasa Polinom. Am folosit urmatorul rationament: am creat polinoame folosindu-ma de string-urile primite ca parametri, dupa aceea am calculat operatia respectiva, rezultatul memorandu-l in polinoamele rezultat si, la final, am schimbat etichetele corepsunzatoare rezultatelor din interfata view cu rezultatele obtinute.

4.5 Controller

Controller-ul primeste ca parametri in constructor un view si un model. Pe langa aceste campuri, mai dispune si de un string, in care este memorata operatia. De asemenea, tot in contructor am si creat ActionListener-i pentru fiecare buton in parte, pe care i-am implementat mai departe ca clase care implementeaza interfata ActionListener. Acolo le-am descris si comportamentul, care bazeaza, in mare, pe urmatorii pasi, la operatii: sunt schimbate etichetele de la rezultat si opTitle din view in functie de operatia selectata, string-ului operation ii este dat valoarea operatiei selectate, se adauga sau se sterg din panel-ul panelIO operanzi si rezultate in functie de operatia curenta si la final etichetele de rezultat sunt curatate, iar frame-ul se redimensioneaza dupa cum s-au modificat elementele lui.

La implementarea lui CalcListener (butonului de calcul) se verifica prima data daca a fost selectata o operatie (cazul in care se afiseaza mesaj de eroare), iar daca asa este atunci se apeleaza operatia corespunzatoare din model cu parametri operanzii extrasi din campurile de input din interfata (String-uri).

1. Rezultate

Pentru fiecare operatie in parte pe polinoame am folosit testarea cu JUnit pentru a fi mai usor de verificat corectitudinea metodelor. Am dat cateva exemple de polinoame cu care am efectuat operatii. Am folosit assertTrue in care am verificat ca rezultatul primit in urma apelarii metodelor sa fie identic cu ceea ce m-am gandit eu ca ar trebui sa fie, adica rezultatul calculului facut de mana. Acest mod de a ma verifica m-a ajutat foarte mult fiind mai rapid decat de a da mereu 2 polinoame sau unul singur, in functie de caz, si compara vizual polinomul cu un altul.

1. Concluzii

Acest proiect mi-a aprofundat cunostintele in Java si, fiind mai amplu, m-a invatat cum sa imi structurez munca mai bine si mai eficient. Pe langa toate acestea, am lucrat cu arhitectura MVC, pe care nu am mai folosit-o asa aprofundat pana in acest moment. A fost un proiect complex care m-a dezvoltat pe mai multe planuri.

1. Bibliografie

* <https://www.tutorialspoint.com/index.htm>
* <https://www.w3schools.com/>
* <https://www.javatpoint.com/>