DOCUMENTATIE

TEMA 1

NUME STUDENT: GHERMAN COSMIN-NICOLAE
GRUPA: 30228

CUPRINS

1. Obiectivul temei

Această temă are obiectivul principal de a aprufunda noțiunile deja dobândite la Programarea Orientată pe Obiecte, prin implementarea unui calculator polinomial la care se adaugă un design realizat printr-o interfață grafică, utilă pentru o navigare mai ușoară a utilizatorului în utilizarea codului. Aceasta team prevede și o serie de obiective secundare dintre care eu le-am remarcat pe acestea:

- Folosirea codului curat, scris cu multe comentarii si cat mai lizibil si ușor de înțeles
- Învatarea utilizării HashMap-ului
- Acomodarea cu buclele foreach
- Crearea unei interfete grafice fară a abuza de “drag-and-drop”, ci folosind mai mult anumite layout-uri predefinite
- Implementarea de metode
- Testarea metodelor cu Junit
- Folosirea regex-urilor pentru parsarea unui string

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

Pentru început utilizatorul ar fi nevoit să introducă de la tastatură cele două polinoame care ar putea să fie o parte destul de dificilă pentru că acestea trebuie introduse după un anumit format. Desigur, acel format trebuie să fie cât mai permisiv cu utilizatorul, dar în același timp trebuie să fie foarte riguroasă pentru a nu duce la greșeli de calcul. Aducând vorba de input, automat rezultă că va fi nevoie de o parsare a acestui string într-un polinom.

În momentul în care utilizatorul apasă un buton, aceasta parsare este realizata in cadrul metodelor implementate in spatele butoanelor. Parsarea are doua lucruri de făcut: să verifice dacă textul introdus poate fi descompus corect în monoame (de exemplu: “*1x^1+1”*, contraexemplu: “*aaaa”)* și transformarea efectivă a string-ului in monoame si ulterior monoamele adăugate in polinom (dacă se va putea). Există posibilitatea ca după parsare să se ajungă la o eroare și pentru asta putem arunca o excepție care va fi prinsă de către metoda care implementează această parte de cod.

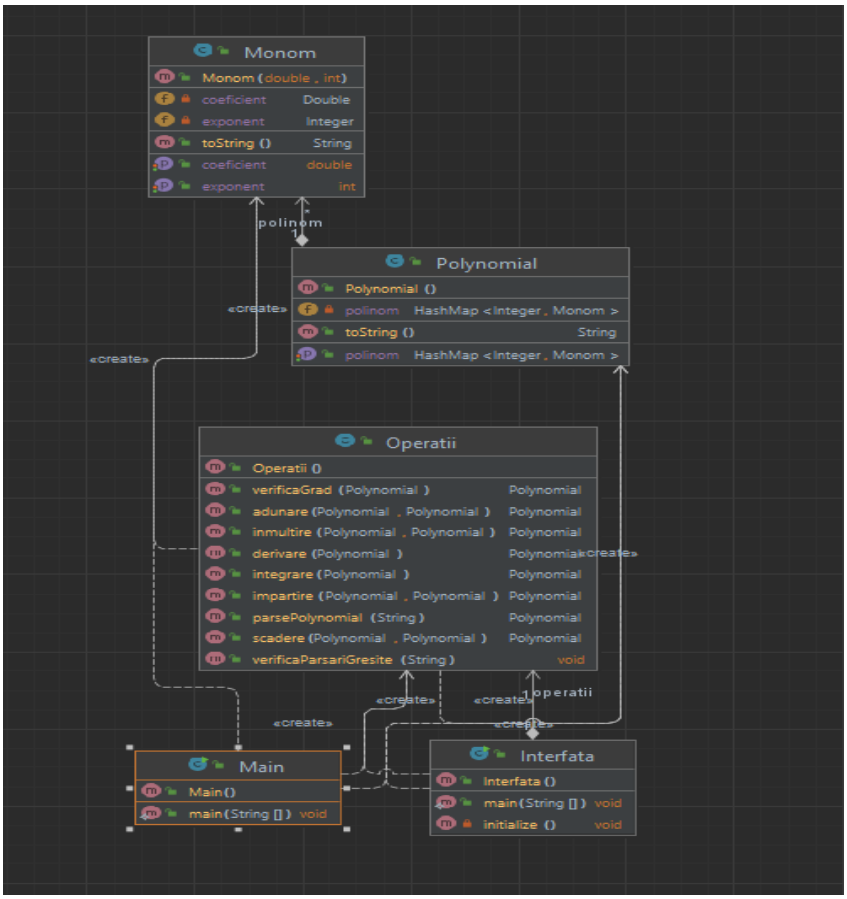
Dacă s-a trecut de etapa parsării, ar urma etapa de operații. Operațiile pot fi de adunare, scădere, înmulțire, împărțire, derivare și integrare. În viitoarea implementare a acestor metode trebuie ținut cont de semne, exponenții și coeficienții monoamelor și mai ales de monomul care are coeficientul și exponentul egal cu 0. În cazul înmulțirii cu 0 automat rezultatul trebuie sa fie 0, în cazul derivării unui coeficient al

lui x^0 rezultatul trebuie sa fie si de aceasta data 0. Pentru derivare și integrare vor trebui luate in calcul neapărat formulele de calcul matematic al unui polinom.

În momentul în care se termină operațiile, utilizatorului i se va furniza pe ecran rezultatul operației dorite între cele două polinoame date de la tastatură.

3. Proiectare

- *Interfață grafică*



- *Diagramă UML de clase*

La nivel de proiectare trebuie specificată importanța fiecărei clase din viitoarea implementare. Asfel cele două clase Monom și Polynomial sunt baza acestui proiect. Dintre aceste două clase, clasa Monom va fi cea care va fi prima implementată deoarece in structura clasei Polynomial se va folosi o structură de HashMap care va contine un Integer ca o cheie și un Monom ca o valoare. Din punct de vedere al eficientei, TreeMap-ul era mai intuitiv de folosit deoarece avea optiunea de a sorta după gradul exponentului rezultatul sau chiar polinoamele ce erau stocate, dar HashMap-ul este de asemenea, o foarte bună alegere. Din punct de vedere al complexității HashMap-ul are o complexitate mai rea decât TreeMap-ul (O(n) față de O(logn)), dar tind să cred că este mai ușor de folosit.

Pe baza acestor două clase fundamentale ale proiectului, se va dezvolta clasa Operatii. În această clasă vor urma să fie implementate toate operațiile. În plus, cu siguranță aici va exista si metoda de parsare a string-urilor cât și o metodă de verificare a gradului fiecărui monom pentru a nu exista un polinom cu monoame cu exponenți identici. Se vor implementa cu siguranță următoarele metode: adunare, scădere, înmulțire, împărțire (nu a fost implementată), derivare și integrare. Toate aceste operații vor primi două polinoame ca argumente, în afară de derivare și integrare care vor primi doar un singur polinom (automat dacă rezultatul este corect pentru un polinom ar putea fi corect și pentru celălalt).

Toate aceste clase se vor îmbina în clasa Interfata. În aceasta clasă am de gând să utilizez un BorderLayout pentru a împărți pagina în 3 regiuni. Aici cu ajutorul butoanelor, textField-urilor, label-

urilor, voi demonstra funcționalitatea celor 5-6 operații aritmetice. Pe lângă aceste detalii tehnice ale interfeței, o să încerc sa proiectez un design plăcut ochiului, care să îl ajute pe utilizator.

4. Implementare

Clasa Monom are două variabile de instanță private : coeficient de tip Double și exponent de tip Integer. Aceste field-uri find private se vor face vizibile către alte clase cu ajutorul metodelor de getter si setter. Această clasă are un constructor care primeste două argumente, un coeficient și un exponent, și inițializează variabilele instanță ale obiectului creat. Se ajunge astfel la metoda suprascrisă toString() acolo unde se verifică mai multe cazuri ale coeficienților si al exponenților pentru a returna un șir cât mai bine scris. Aceste if-uri sunt foarte ușor de înțeles si urmărit ceea din face din această metodă sa fie foarte ușor de înțeles.

Clasa Polynomial este următorul pas dupa clasa Monom. Această clasă reprezintă un polinom si conține un HashMap de obiecte Monom care reprezinta termenii polinomului. HashMap-ul polinomului conține si o cheie (Integer-ul) care este de fapt exponentul monomului. Metodele implenetate aici sunt : constructorul Polynomial care nu are argumente si initializează obictul cu un HashMap nou pentru polinom, getter-e si setter-e pentru polinom, si metoda toString(), metoda care returnează o reprezentare sub formă de sir a polinomului, prin concatenarea șirurilor obtinute prin apelarea metodei toString() a fiecarui obiect Monom din polinom. Acel if din acea metoda verifica daca polinomul este gol (adică dacă nu continue monomi) si returnează 0.

Clasa Operatii are rolul de a realiza operațiile ce se pot realiza între două polinoame. În această clasa se observă folosirea obiectelor de tip Polynomial si Monom pentru calcularea operațiilor. Astfel clasa are metodele de:

- *verificaGrad(Polynomial p1)* -> este o metoda prin care se transmite un obiect p1 de tip Polynomial si este returnat un nou obicet de tip Polynomial ca rezultat. Singurul scop al acestei metode este de a simplifica input-ul, adica in momentul in care exista doua monoame cu acelasi exponent intr-un polinom, sa le adune coeficientii inainte de a incepe o operatie propriu zisa. Rezultatul acestei simplificari este ulterior transmis catre functia care va face operatia ceruta.
- *adunare(Polynomial p1, Polynomial p2)* -> aceasta metoda primeste doua polinoame ca input si ofera un rezultat de tipul Polynomial ca return. Prima data in aceasta metoda se verifica cu ajutorul metodei *verificaGrad(Polynomial p1)* daca avem ceva de simplificat (aceasta metoda va fi aplicata in fiecare functie si o sa precizez rolul ei doar aici). Dupa aceasta cream rezultatul pe care am zis ca o sa il returnam. Cu doua foreach-uri parcurgem monom cu monom cele doua polinoame si adunam coeficientii monoamelor care au acelasi exponent. Ne vom folosi de un monom auxiliar pentru a seta noul coeficient si noul exponent al monomului rezultat, care va fi adaugat in polinomul rezultat. In cazul in care exista o valoare unica a unui exponent in polinomul 1 atunci aceasta se va insera in rezultat, urman ca dupa terminarea celor doua for-uri sa mai executam un for pentru a parcurge cel de-al doilea ploinom si sa verificam daca rezultatul nu contine deja exponentul cautat, iar daca nu, introducem monomul in rezultat. Verificam gradul rezultatului pentru posibile simplificari si ulterior returnam rezultatul.
- *scadere(Polynomial p1, Polynomial p2)* -> operatia de scadere a fost foarte usor de implementat avand in vedere utilitatea metodei de adunare. Pentru a face o scadere am parcurs fiecare monom

al polinomului2 si pentru coeficientii acestuia am schimabt semnul. Dupa ce am schimbat semnul folosindu-ne din nou de un monom auxiliar, facem operatia de adunare intre polinomul1 si inversulPolinomului2 care se va sctoca intr-un prolinom rezultat care la randului lui va fi returnat.

- *inmultire (Polynomial p1, Polynomial p2) ->* pentru aceasta medota am parcurs cele doua polonoame monom cu monom si am inmultit coeficientul primului monom cu coeficientul celui de al doilea monom, la fel si pentru exponent, si am stocat totul intr-un monom auxiliar, care a fost adaugat la polinomul rezultat dupa fiecare trecere prin bucla for. Aici a trebuit adaugata conditia ca in momentul in care avem un coeficient 0, monomul rezulat sa fie 0. In cazul in care un monom din rezultat are exponentul egal cu exponentul din auxiliar, se vor aduna coeficientii celor doi si se va pastra puterea. Auxiliarul va fi pus in result care va fi returnat.
- *impartirea (Polynomial p1, Polynomial p2)->* aceasta operatie nu am implementat-o
- *derivare (Polynomial p1) ->* Aceasta medota primeste doar un polinom si are o singura formula de aplicat, in mometul in care monomul are exponentul diferit de 0 sa inmulteasca coeficeientul cu exponentul si sa seteze coeficientul auxiliarului cu acest rezultat si exponentul auxiliarului va fie egal cu exponentul monomului -1. In cazul in care exponentul monomului este 0, atunci inseamna ca rezultatul derivarii trebuie sa fie 0. Aceasta implementarea s-a realizat cu o bulca foreach si pentru fiecare monom s-au facut verificarile spuse anterior.Derivata din 0 este 0. Monomul auxiliar este pus in polinomul rezultat care este si returnat la final.
- *integrarea (Polynomial p1) ->* Si aici ca la derivare se urmareste tot o formula. Aici de fiecare data exponentul monomului auxiliar va primi exponentul monomului +1 iar coeficientul auxiliarului este raportul dintre coeficientul actual si exponentul incrementat cu 1 al monomului. Integrala din 0 este 0. Monomul auxiliar este pus in polinomul rezultat cate este si returnat la final.
- *verificaParsariGresite(String input) ->* primește un sir de caractere "input" și verifica daca acesta conține caractere nepermise. Daca exista astfel de caractere, metoda aruca o exceptie cu mesajul "Pattern gresit!". Metoda este apelata la începutul metodei "parsePolynomial" pentru a se asigura ca sirul de intrare este corect.
- *parsePolynomyal(String input) ->* primește un șir de caractere "input" și creează un polinom corespunzător. Mai întâi, se verifică dacă șirul de intrare este corect folosind metoda "verificaParsariGresite". Apoi, se definește o expresie regulată pentru a extrage fiecare monom din șirul de intrare. Monomurile sunt adăugate la polinom folosind un ciclu "while". Fiecare monom este procesat pentru a se obține coeficientul și exponentul și apoi se adaugă la polinom. Dacă polinomul conține deja un monom cu același exponent, coeficientul noului monom este adăugat la cel al monomului existent.

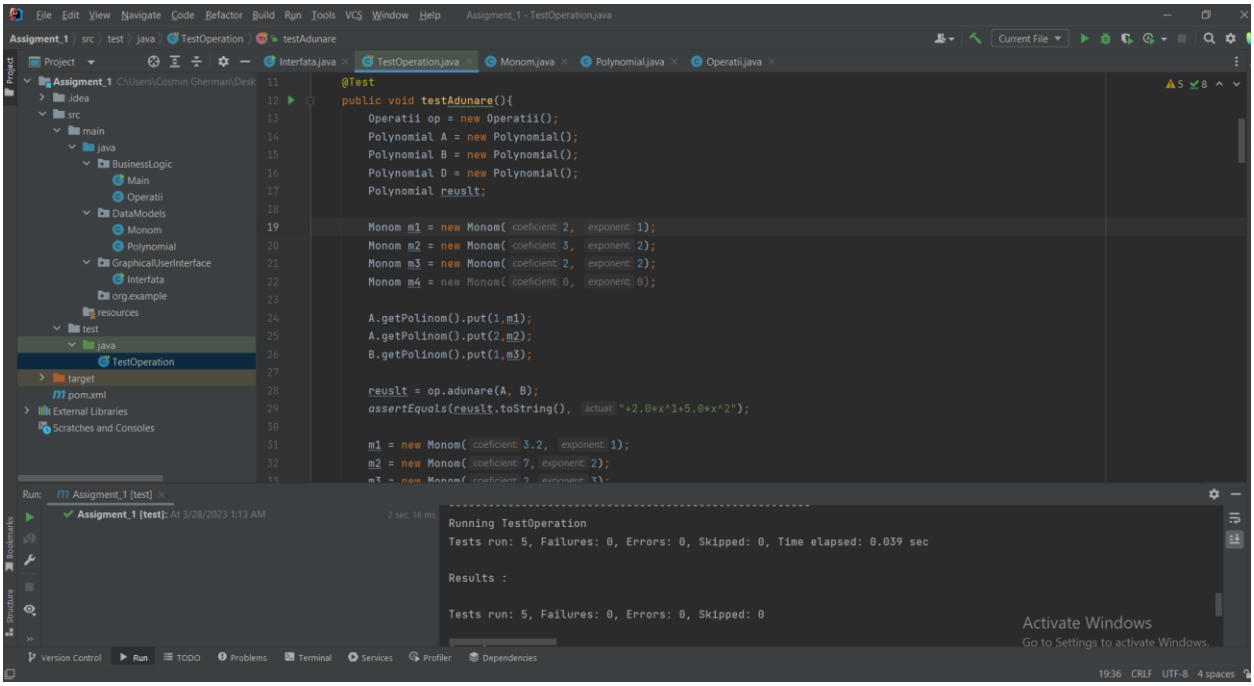
Clasa Interfata este reprezentata de o interfata grafica cu utilizatorul (GUI). Aceasta permite introducerea celor doua polinoame in doua textField-uri si permite afisarea lor intr-un textArea. Aceste field-uri au inaintea lor niste label-uri pentru a ilistra exact ce sa se introduca in textField-uri. Aceasta interfata este facuta cu BorderLayout si este inpartita in 3 mari zone : nord, centru si vest. In partea de nord exista textfield-urile, label-urile cu titlul, exemplul de polinom si denumirile textfield-urilor si doua butoane care au rolul de a sterge ce se afla in interiorul polinoamelor. In partea din mijloc sau partea centrala a interfetei sunt butoanele principale, care au scopul de a face operatiile dintre polinoame. Fiecare buton din aceasta interfata are implementata o functie care sa-I perimta sa primeasca comenzi de la utilizator, In partea de sud, se remarca doua butoane care ofera utilizatorului

intr-un pop-up exemple de polinoame pe care daca vrea sa le introduca, acestea se vor seta direct in textfield-uri. Aici polinoamele date exepluc sunt retinute ca stringuri si oferite catre texfield-uri. Fiecarui buton, in momentul in care polinomul introdus nu este dupa forma ceruta, i-am implementat o exceptie care arunca prin intermediul unor pop-up-uri mesaje de eroare si de atentionare.

In clasa main au avut loc toate testele operatiilor inainte de a crea interfata si inainte de a face testarea cu JUnit. In clasa de teste, am create metode de teste pentru fiecare operatie si folosind assertequals am verificat daca rezultatul oferit ca string coincide cu string-ul la care ma asteptam eu sa fie.

5. Rezultate

Pentru fiecare operatie implementata in clasa Operatii am realizat o functie de testare. In aceasta functie de testare am procedat exact ca pentru testarea in main, doar ca la final in loc sa verific in consola rezultatul daca coincide cu asteptarile mele, am folosit metoda assertequals care imi returna un mesaj de true in finalul testului daca string-ul rezulatului coincide cu string-ul introdus de mine. Mai jos se poate remarca cum toate metodele impementate au trecut cu succès testele, si se poate observa si metoda din testAdunare.



6. Concluzii

Concluzionez prin a spune ca aceasta tema mi-a imbunatatit nivelul de intelegere al folosirii pachetelor, al folosirii Map-ului, foreach-ului. Aceasta implementare a calculatorului polinomial poate fi ulterior dezvoltata si perfectionata pentru a putea face parte din calculatorul unui teleofn spre exemplu sau a unui sistem de calcul mult mai complex care nu se bazeaza doar pe polinoame ci si pe alte expresii matematice (de exepmlu PhotoMath).

7. Bibliografie

1. *https://regexr.com/ -> pentru a testa expresia regulate*
2. *https://stackoverflow.com/questions/20496239/maven-plugins-can-not-be-found-in-intellij -> pentru rezolvarea reorilor legate de plugin-uri*
3. *https://stackoverflow.com/questions/68827392/parsing-a-string-using-java-regex ->despre parsare*
4. *https://dsrl.eu/courses/pt/materials/PT2023_A1_S1.pdf -> using map*
5. *https://app.diagrams.net/ -> pentru diagrama de clase*