DOCUMENTAȚIE

TEMA *#1*

NUME STUDENT: TRUȚA ANDREI
GRUPA: 30221

CUPRINS

1. Obiectivul temei

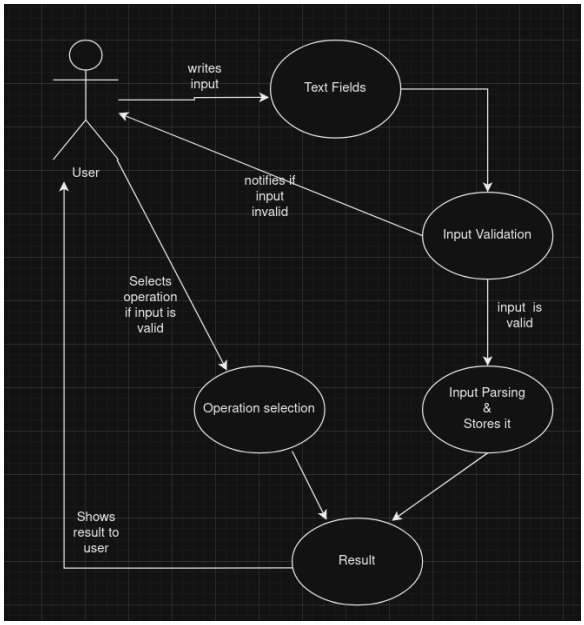
Obiectivul principal al acestei teme este dezvoltarea unei aplicații interactive pentru calcule asupra polinoamelor. Pentru realizarea acestei teme au fost urmărite câteva etape intermediare:

- *Determinarea nevoilor utilizatorului – o scurtă analiză referitoare la cerințele funcționale ale aplicației, precum modul de interacțiune cu utilizatorul (cap. 2)*
- *Dezvoltarea unei interfețe grafice pentru interacțiunea cu utilizatorul (cap. 4)*
- *Proiectarea unui flow de date, adică transmiterea input-ului utilizatorului prin nivelele aplicației cu scopul de procesare și validare (cap. 2)*
- *Dezvoltarea unui design logic al aplicației, adică clasele necesare, relațiile între acestea și interacțiunile lor (cap. 3)*
- *Implementarea propriu-zisă a claselor și dezvoltarea algoritmilor pentru operațiile pe polinoame (cap. 3)*
- *Îmbunătățirea rigidității aplicației cu ajutorul testelor unitare (cap. 5)*

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

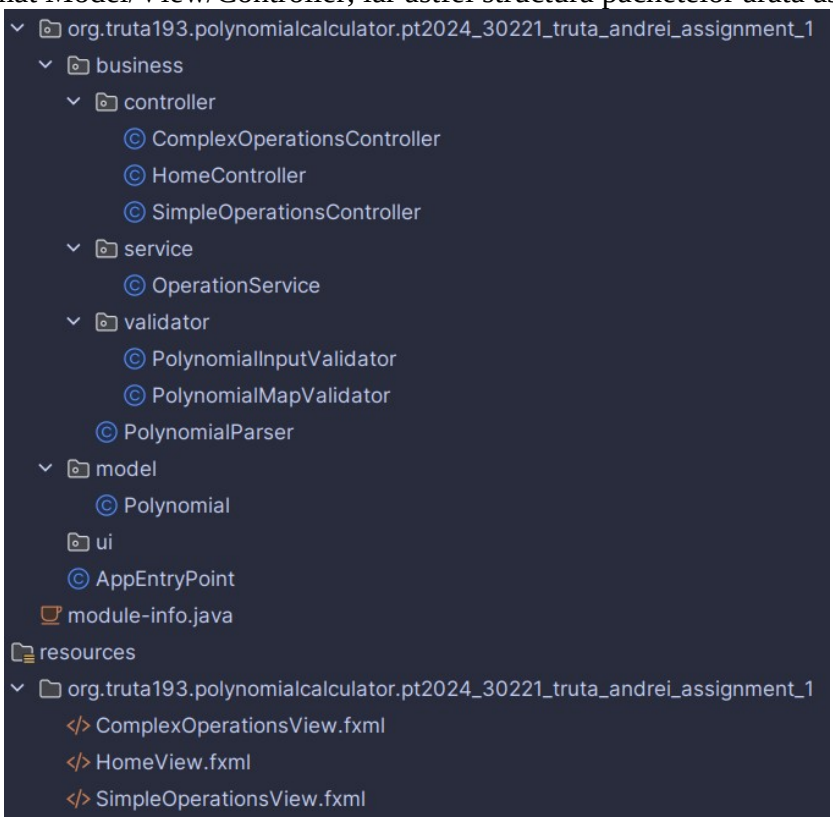
Cerințele funcționale ale aplicației ne descriu modul în care aceasta trebuie să funcționeze. Pentru început, avem nevoie de o metodă de preluare de date de la utilizator, lucru realizat printr-un TextField în interfața grafică a aplicației. Datele sunt validate și prelucrate iar mai apoi, dacă acestea sunt coerente, este oferit un set de operații asupra polinoamelor. Se va returna utilizatorului rezultatul corect al operației.

Pentru partea de cerințe non-funcționale, aplicația arată erori utilizatorului când o dată de intrare este validă și blochează continuarea flow-ului. De asemenea, interfața este structurată în două taburi, unul pentru operații „simple”, adică cele clasice, pe două polinoame, și un tab pentru operații „complexe” adică derivarea și împărțirea, ce necesită un singur polinom.



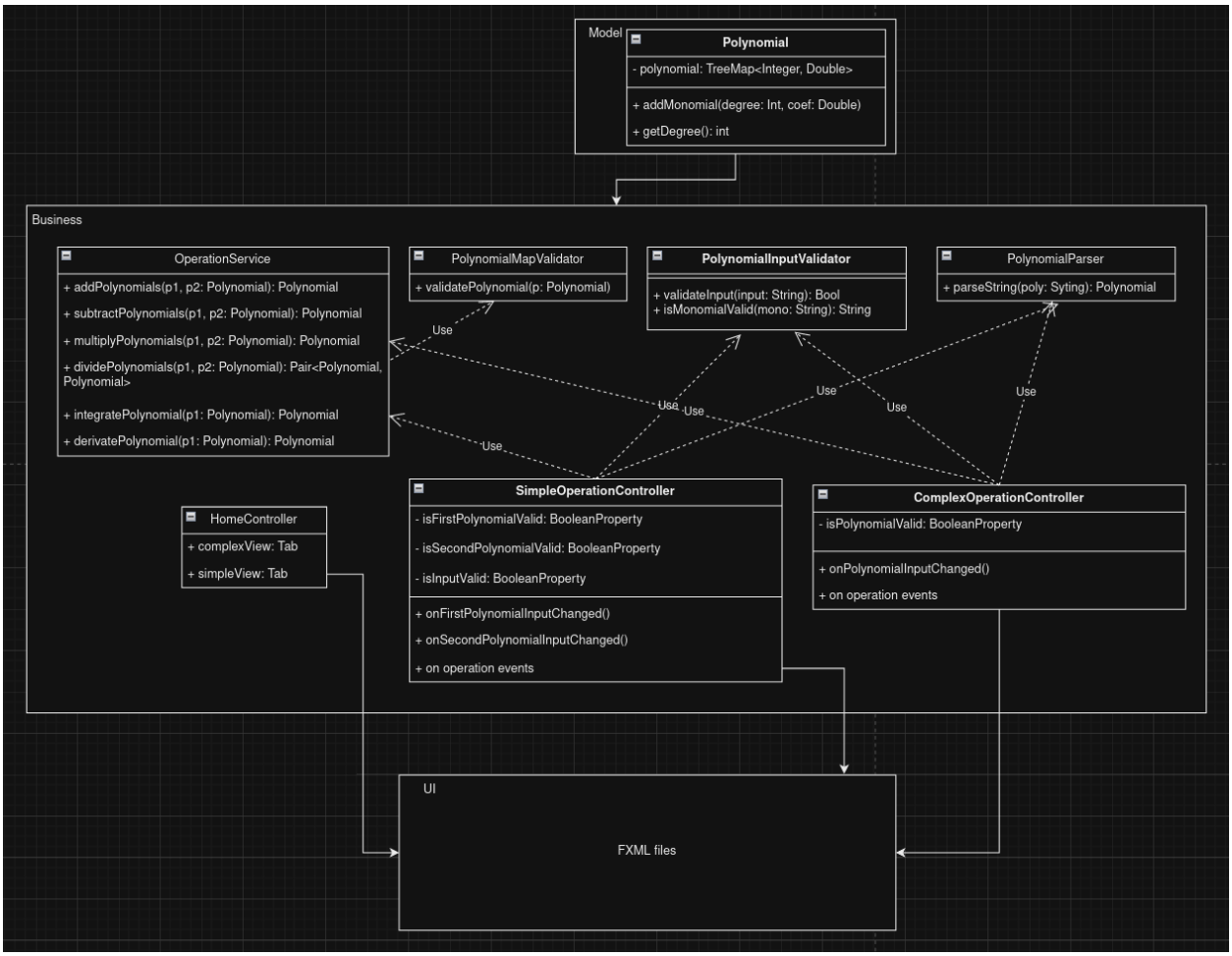
3. Proiectare

În cadrul aplicației am urmat structura de Business/Presentation/Model, iar pentru partea de interfață, am urmat Model/View/Controller, iar astfel structura pachetelor arată astfel:



Structura de date folosită pentru reprezentarea polinomului este un TreeMap. Am ales această structură deoarece operațiile pe ea, deşi nu la fel de rapide ca un HashMap (O(lgn) vs O(1)), pastrează o relație de ordine a termenilor datorită implementării sale cu un Red-Black Tree. Astfel, puterile sunt stocate in ordine crescătoare, iar clasa permite parcurgerea lor şi în ordine inversă, lucru necesar pentru implementarea derivării.

Mai jos este prezentată diagrama UML pentru aplicație, unde se poate vedea foarte evident faptul că am ales să structurez proiectul în clase concrete cu unic scop.



4. Implementare

Clasele importante care merită menționate și detaliate :

- *Polynomial*
  - *Câmpul polynomial de tipul TreeMap<Integer, Double> este modalitatea prin care se stochează polinomul; cheia este puterea, aceasta fiind unică într-un polinom, iar valoarea este coeficientul termenului respectiv. Precum s-a discutat în cap. 3, decizia de a folosi TreeMap se datorează uneia dintre cerințele temei, aceea de a afișa polinomul de la putere mare la putere mică, descrescător. Deși toate operațiile au complexitatea în timp O(lgn), avându-le deja sortate, scăpăm de pasul de sortare de la afișare, care este O(nlgn).*
  - *Metoda addMonomial are ca scop adăugarea unui termen în polinom. În cazul în care termenul există deja, se vor aduna coeficienții.*

- *Metoda getDegree are ca scop determinarea gradului polinomului (cea mai mare putere cu coeficient non-zero).*

- *PolynomialMapValidator*
  - *Metoda validatePolynomial are ca scop validarea acestuia prin eliminarea termenilor cu coeficient 0. Deși aceștia nu încurcă la realizarea operațiilor, afișarea este mai frumoasă cu acești termeni scoși.*
- *PolynomialInputValidator*
  - *Metoda isMonomialValid folosește pattern matching să verifice dacă un termen al polinomului este valid d.p.d.v sintactic, cu ajutorul expresiei* ([+-]?\\d*)(x(\\^\\d+)?)?:
    - ([+-]?\\d*) - grup pentru coeficient, poate să existe sau nu, cu sau fară semn.
    - (\\^\\d+)? - grup pentru putere, poate să existe sau nu, este opțional deoarece x de gradul 1 nu are nevoie sa ii se specifice faptul că este la puterea 1.
    - (x(\\^\\d+)?)? - grup pentru x la o putere, termenul liber nu are x, de aceea acest grup este optional.
  - Metoda validateInput are ca scop determinarea corectitudinii întregului polinom de intrare. Realizează acest lucru făcând un split după semne și apelând isMonomialValid pe fiecare substring.
- PolynomialParser
  - Metoda parseString se folosește de același regex ca mai sus, împreună cu un Matcher, pentru a prelucra string-ul de intrare (în momentul în care string-ul ajunge la acest pas, este deva validat sintactic) transformându-l într-ul Polynomial.
- OperationService
  - Metoda addPolynomials are ca rol adunarea a două polinoame.
  - Metoda subtractPolynomials are ca rol scăderea a două polinoame.
  - Metoda multiplyPolynomials are ca rol înmulțirea a două polinoame.
  - Metoda dividePolynomials are ca rol împărțirea a două polinoame.
  - Metoda integratePolynomial are ca rol integrarea unui polinom.
  - Metoda derivatePolynomial are ca rol derivarea unui polinom.
- SimpleOperationsController
  - Clasă care se ocupă de comunicarea cu layer-ul de prezentare; aici se face legătura între acțiunile utilizatorului (scris date de intrare, apăsări de butoane) cu logica aplicației prin intermediul evenimentelor de tip callback.
- ComplexOperationsController
  - Analog cu SimpleOperationsController, diferența fiind că există un singur câmp pentru date de intrare aici.

5. Rezultate

Testarea a fost realizată atât manual cât și cu ajutorul testelot unitare prin JUnit. Testarea unitară a avut ca scop principal menținerea rigidității și a corectitudinii aplicației pe parcursul modificărilor acesteia, iar accentul s-a pus pe operații (comportamentul acestora la date de

intrare mai speciale, precum polinom NULL sau împărțire la 0) cât și pe validarea regex a polinoamelor. Din 26 de teste scrise la finalul proiectului toate trec.

6. Concluzii

Aplicația a ajuns într-un stadiu favorabil de funcționare, iar pe parcursul dezvoltării acesteia am învățat să lucrez cu regular expressions și mi-am dezvoltat cunoștințele de structurare a unui proiect în module cu scop specific pentru ușurarea muncii, mai ales când o componentă trebuie schimbată. Ca și dezvoltări ulterioare, înterfața necesită îmbunătățiri, poate chiar refacerea acesteia cu o bibliotecă mai modernă, sau continuarea cu JavaFX si aducerea design-ului la un standard precum Material3. O altă idee ar fi introducerea unei clase numită Monom și folosirea acesteia pentru a stoca coeficientul sub formă simbolică în loc de numerică, precum este acum implementat. Afișarea „1/3” este mai precisă și frumoasă decat „0.333...”.

7. Bibliografie

1. *Division Algorithm for Polynomials - https://www.geeksforgeeks.org/division-algorithm-for-polynomials/*
2. *Cheatsheet - https://regexr.com/*
3. *MVC Design Pattern - https://www.geeksforgeeks.org/mvc-design-pattern/*
4. *JavaFX FXML - https://jenkov.com/tutorials/javafx/fxml.html*
5. *Regex in Java - https://www.w3schools.com/java/java_regex.asp*