MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Facultatea de Automatică și Calculatoare
Departamentul Calculatoare

TEMA 1

la disciplina

TEHNICI DE PROGRAMARE

- CALCULATOR POLINOMIAL -

Proiect realizat de : Moloce Sabina-Maria

An: 2

Grupa: 30225

ANUL ȘCOLAR 2020-2021

Cuprins

1. Obiectivul temei

1.1 Obiectivul principal

Obiectivul acestei teme de laborator este să proiectăm și să implementăm un program Java specializat pe procesare polinoamelor. Polinoamele sunt de o singură variabilă cu coeficienți întregi.

1.2 Obiectivele secundare

- Dezvoltarea de use case-uri – capitolul 2

Use case-urile reprezintă un set de scenarii legate de modul în care este utilizat sistemul. Utilizarea acestora ne ajută să descoperim:
- entități de sistem
- actori de sistem (roluri)
- atribute
- comportament
- cum interacționează actorii cu resursele sistemului

- Dezvoltarea de diagrame UML pentru pachete și clase – capitolul 3

Vom utiliza diagrama de pachete pentru a asigura arhitectura MVC(Model – View – Controller). Cele trei parți ale acestei arhitecturi fiind separate în câte un pachet. Vom utiliza diagramele UML de obiecte pentru a obține clasele necesare pentru rezolvarea obiectivului principal.

- Dezvoltarea algoritmilor – capitolul 3

Pentru a ne atinge obiectivul principal, avem nevoie să dezoltam algoritmi sau să căutam algoritmi ce realizeaza funcționalitatea dorită. În acest caz, avem nevoie de algoritmi pentru opearțiile pe polinoame.

- Implementarea soluției – capitolul 4

Se va descrie fiecare clasă cu câmpuri și metodele importante. Se va descrie implemantarea interfeței utilizator.

- Testarea – capitolul 5

Vom prezenta scenarii pentru testarea oparațiile pe polinoame.

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

Se dorește o soluție capabilă să efectueze corect următoarele operații: adunarea, scaderea, înmulțirea, împarțirea, derivarea și integrare, dar și care interceptează erorile care pot veni din partea utilizatorului, precum introducerea greșită a datelor de intrare.

Modelarea problemei este, de asemenea, un factor important pentru a ajunge la soluția dorită.

Caz de utilizare: realizarea unei operații pe polinoame

Actor principal: utilizator calculator

Scenariul principal de succes:

1. Utilizatorul introduce în câmpurile corespunzătoare(de exemplu pentru adunare/scadere/înmulțire/împarțire în ambele câmpuri se introduce cate un polinom, iar pentru derivare si integrare neaparăt în primul câmp se face introducere polinomul), polinoame care au o sintaxă corectă.

2. Utilizatorul apasă butonul corespunzător operației dorite

3. Rezultatul este afișat în câmpul de rezultat

Secvențe alternative:

a) Nu au fost introduse polinoamele în câmpurile corespunzătoare

- utilizatorul primește un mesaj care detaliază eroarea și sugerează modul corect de introducere

- utilizatorul introduce din nou polinoamele

b) Sintaxa greșita a polinamelor

- utilizatorul primește un mesaj care-l atenționează că s-au introdus greșit polinoamele

- utilizatorul introduce din nou polinoamele

c) Daca utilizatorul introduce în câmpul 2 polinomul 0 si apasă butonul de impartire

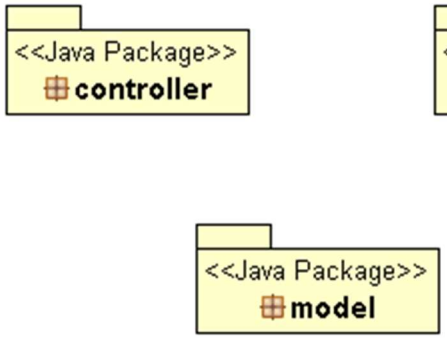
- utilizatorul primește un mesaj care-l anunța ca impartirea cu 0 nu este posibilă

3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfață utilizator)

Am ales sa folosesc arhitectura MVC. Aceasta cere ca o aplicaţie vizuală să fie divizată în trei părţi separate:
- model - reprezintă intern datele aplicaţiei
- vizualizare (view) – reprezentarea vizuală a datelor respective
- controlor (controller) care preia intrarea de la utilizator şi o transpune în schimbări în model

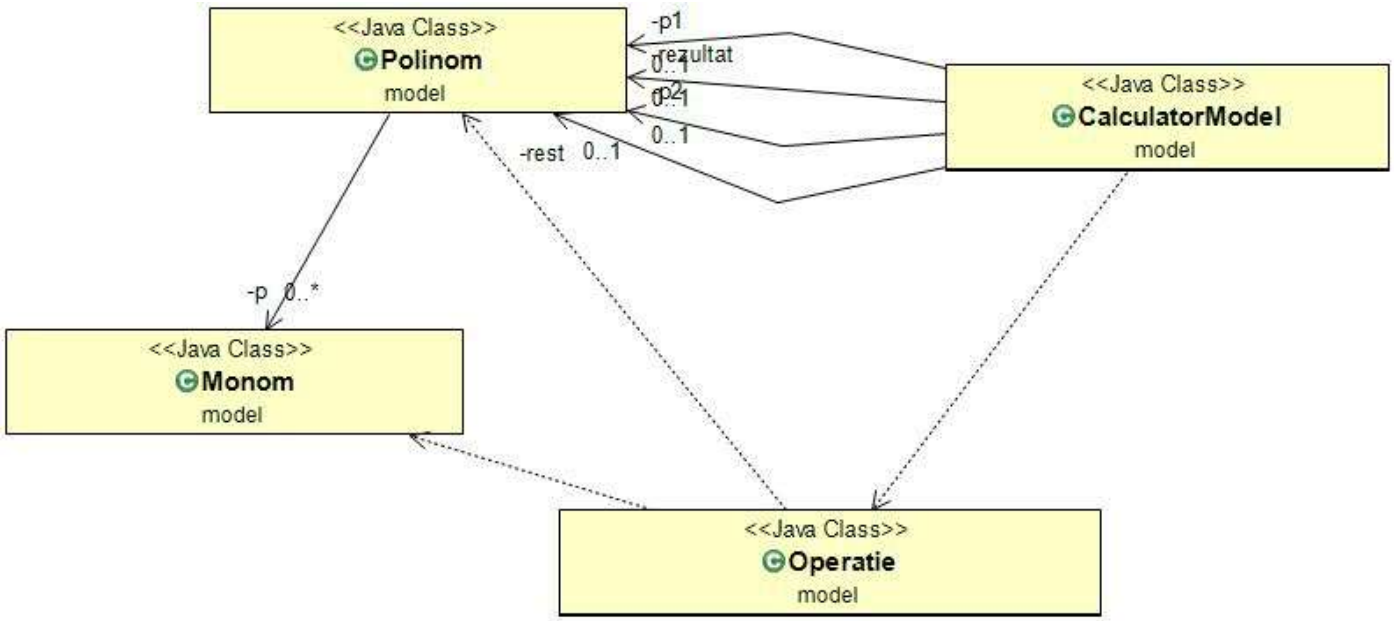
3.1 Diagrame UML

- Diagrama de pachete ne ajută să decompunem sistemul în subsisteme, astfel descompunem sistemul nostru in trei pachete legate strâns de modelul architectural MVC(Model- View- Controller). Deși nu sunt reprezentate în imagine, există relații de asociere între pachetul controller și pachetele view și model.

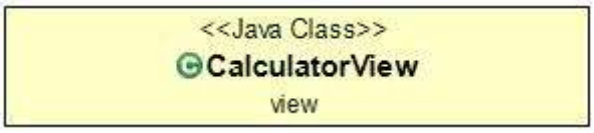


- Diagrama de clase

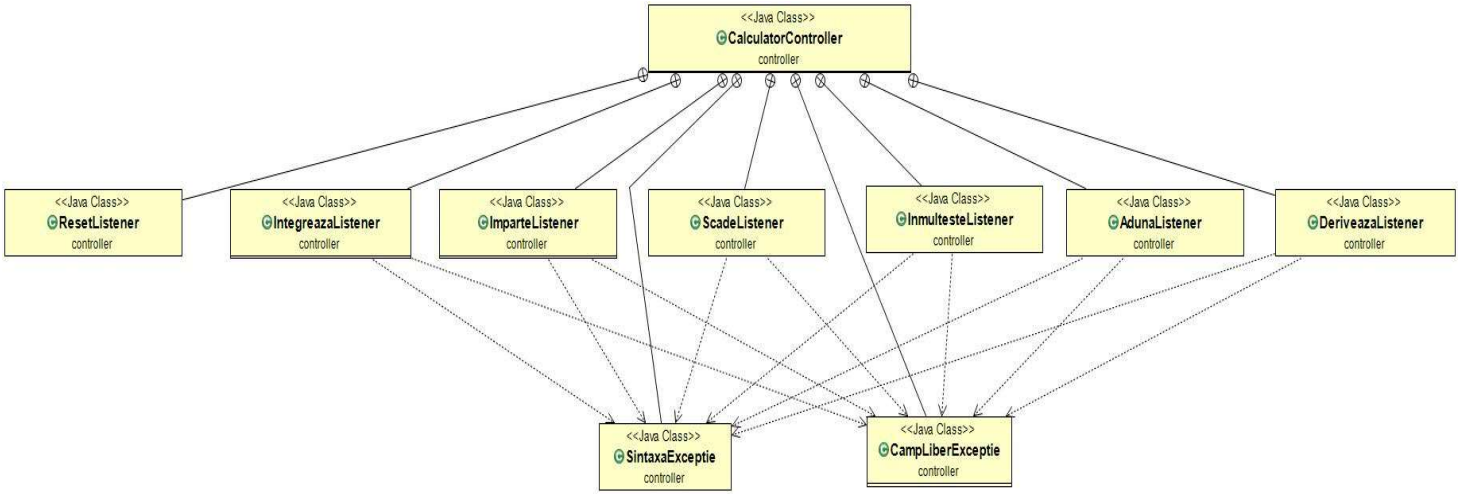
Diagrama de clase descrie clasele din sistem și relațiile dintre ele, ce pot fi asocieri, generalizari, de realizare, de dependenta.
Clasele vor fi doar enumerate în aceast capitol, detaliile despre câmpuri și metode se vor dezvolta în capitolul 5.
Clasele din pachetul model:



Clasa din pachetul view – practic interfața acestei aplicații



Clasa din pachetul controller:



3.2 Structuri de date folosite

In acest proiect singurele structuri folosite sunt listele, implemenatate prin ArrayList. Le folosim la stocarea monoamelor dintr-un polinom.

3.3 Algoritimi utilizați

- Algoritmul pentru împartirea a doua polinoame

function n / d is
  require d ≠ 0
  q ← 0
  r ← n // At each step n = d × q + r

  while r ≠ 0 and degree(r) ≥ degree(d) do
    t ← lead(r) / lead(d) // Divide the leading terms
    q ← q + t
    r ← r − t x d

  return (q, r)

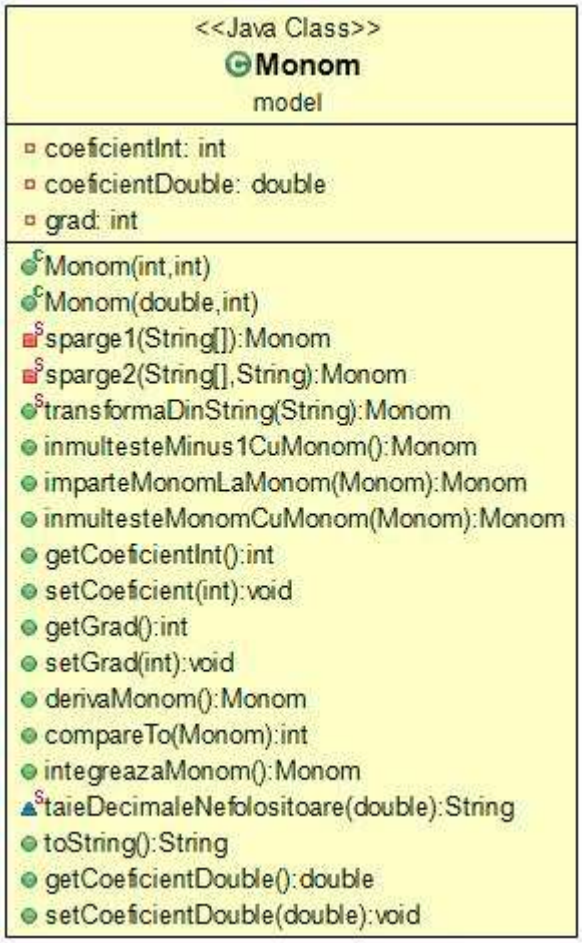
- Formule de derivare si integrare ajutătoare

$$\frac{d}{dx}x^n = n \cdot x^{n-1} \qquad \int x^n\,dx = \frac{x^{n+1}}{n+1} + C$$

4. Implementare

4.1 Clasele din pachetul model

- Clasa Monom



Polinoamele sunt construite din termeni numiți monoame, care sunt alcătuite dintr-o constantă (numită coeficient) înmulțită cu o variabilă. Fiecare variabilă poate avea un exponent constant întreg pozitiv. Exponentul unei variabile dintr-un monom este egal cu gradul acelei variabile în acel monom.Un monom fără variabile se numește monom constant, sau doar constantă. Gradul unui termen constant este 0.

Acestă clasă implementează conceptul matematic de monom.

Câmpurile acestei clase sunt: coeficientInt de tip întreg pe care îl folosim atunci cand citim monoamele de la utilizator, care va introduce datele ca și întregi,; coeficientDouble de tip real pe care îl vom folosi de fapt în calculele noastre, asta pentru ca rezultatul unei impărțiri nu este mereu un întreg, iar la integrarea unui monom avem de realizat impărțiri; grad de tip întreg.

Avem doi constructori ca urmare a faptului ca avem doua reprezentari pentru coeficienti, una intreagă si una reală.

Metodele clasei:
-toString() care returnează monomul sub forma de String. Acesta metoda utilizează metoda taieDecimaleNefolositoare() care primește coeficientul real, iar atunci când primele decimale de după virgulă sunt 0, metoda returnează un String care reprezintă doar partea întreaga din coeficient

-transformaDinString() care primește un String și ne returnează monomul corespunzător. Există mai multe moduri în care utilizatorul poate să dea monomul: “2*x^3”, “2x^3”, “+x^3”, “-x^3”, “2*x”, “2x”, “-x”, “+x”, de aceea avem nevoie de funcțiile auxiliare sparge1 si sparge2 care se ocupă de unele cazuri ce au o rezolavare mai stufoasă.

-metodele integreazaMonom() si derivaMonom() care implementează formulele de la 3.2
-metodele inmulteteMinus1CuMonom() , imparteMonomLaMonom() si inmulteteMonomCuMonom care implementează diverse operații simple cu monome

- Clasa Polinom

Implementăm polinoamele ca liste de monome, iar gradul polinomului este dat de cel mai mare grad pe care îl are un monom.

Metodele clasei:
-stergeMonomeZero() care sterge monamele din polinom care au coeficientul 0, acestea fiind generate in urma oparțiilor de adunare sau de înmultire cu polinomu 0

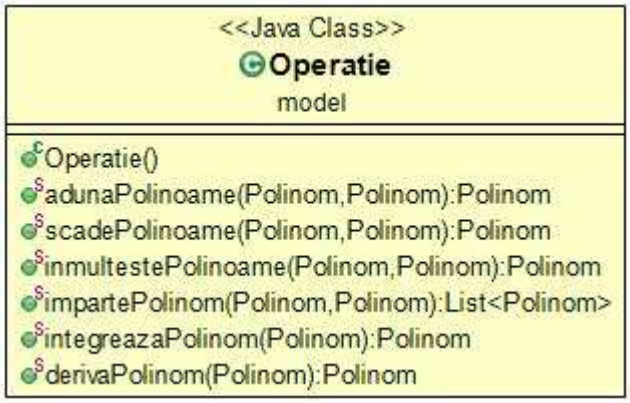
-creeazaPolinomValid() este cea mai importantă funcție din această clasă deoarece parsează un String și dacă acesta nu încalcă sintaxa pentru un polinom, atunci ne returnează un obiect de tipul polinom, altfel returnează null.
Folosim interfața Regular Expression sau Regex care ajută la definirea unui model pentru a căuta și manipula Stringuri.
Definim urmatorul pattern pentru monome și îl compilăm:
([+-]((\\d+[*]?x\^\\d+)|(x\\^\\d+)|(\\d+[*]?x)|(\\d+)|(x)))

Cu ajutorul unui matcher: folosind modelul compilat mai sus, efectuăm operațiunilor de potrivire. Atât timp cât gasim potriviri în

Stringul nostru înseamnă că mai avem grupuri de monome pe care le vom transforma cu metoda transformaDinString(). La sfarșit verificăm daca au rămas caractere care nu au făcut parte din nici un grup, și dacă sunt returnăm null.

- Clasa Operație

Este clasa proriu-zisă unde sunt implementate operațiile principale.



Metodele clasei:
-adunaPolinome(p1, p2): adunăm la rezultat monoamele din polinomul 1 si după adunăm monoamele din polinomul 2, însa înainte verificăm dacă există în rezultat un monom care are același grad, daca gasim atunci adunăm coeficienții.

-scadePolinoame(p1, p2): inmultim fiecare monom din polinomul p2 cu -1, astfel obtinem polinomul -p2, dupa care efectuam adunarea dintre p1 si -p2.

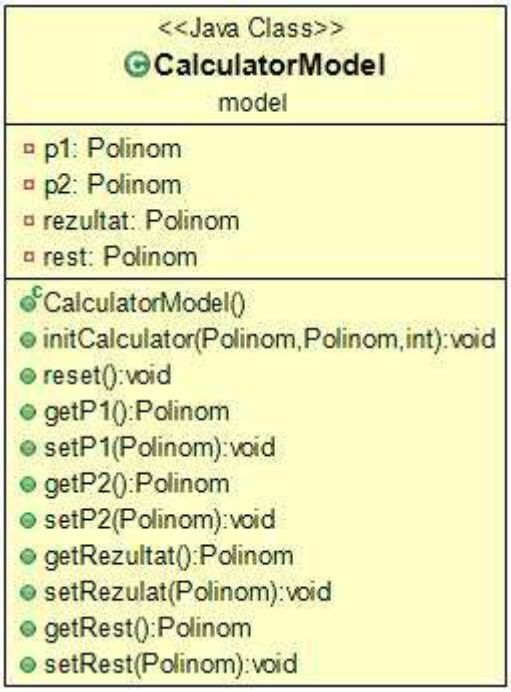
-inmultestePolinoame(p1, p2): înmulțim fiecare monom din polinomul p1 cu polinomul p2 și adunăm rezultatele intermediare pentru a obține rezultatul final.

-impartePolinoame(p1, p2): implementăm pas cu pas algoritmul descris la 3.2

-integreazaPolinom(p1) si derivaPolinom(p1) se bazeaza pe integrarea fiecărui monom din polinom, respectiv pe derivare

- Clasa CalculatorModel

Are ca si câmpuri 4 polinoame: polinomul1, polinomul2, rezultat și rest (pentru împarțire).



Metodele clasei:
-setters și getters pe care îi vom utiliza în controller
-initCalculator(p1, p2. codOperatie) care inițializează valorile polinoamelor p1 și p2 și calulează rezultatul în funcție de codul operației primit:

adunare=0
scadere =1
înmulțire=2
împarțire=3
derivare=4
integrare=5

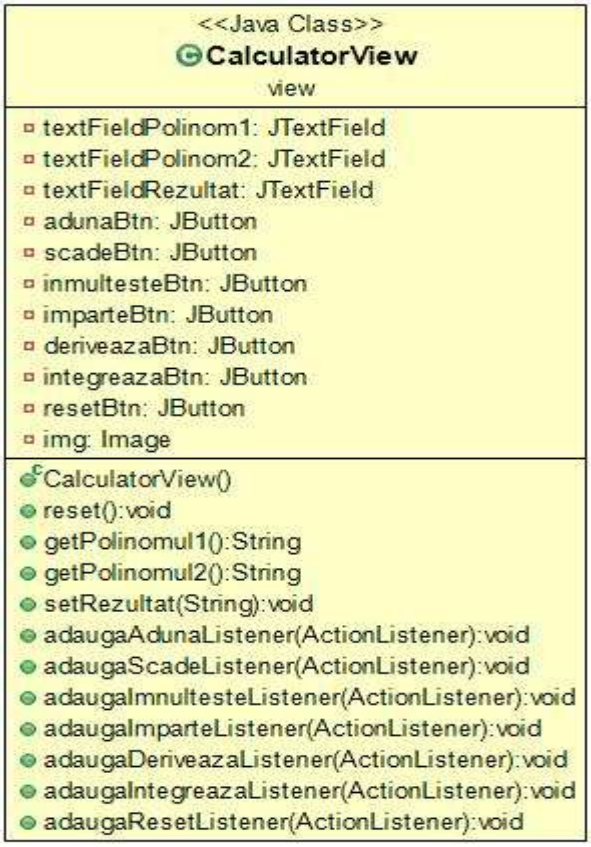
4.2 Clasa din pachetul view

- Clasa CalculatorView

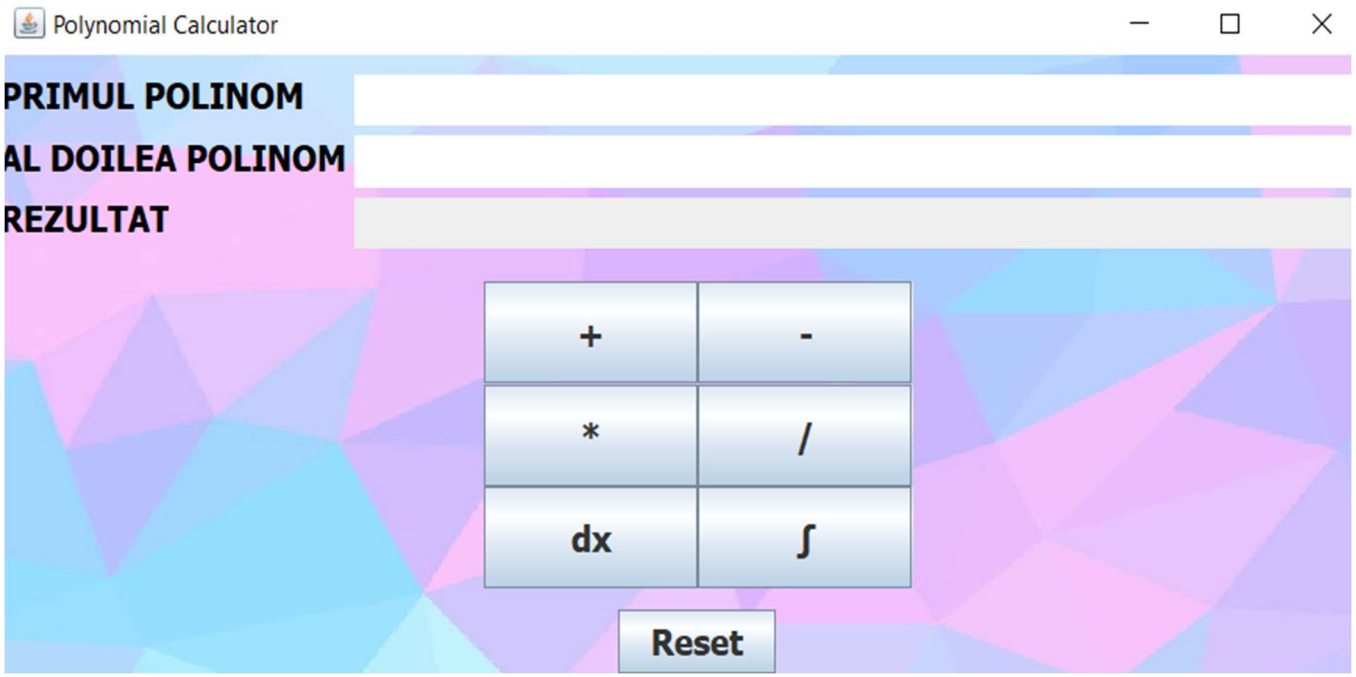
Câmpurile clasei sunt trei componente de tipul JtextField, două pentru a introduce polinoamele și unul care nu este editabil pentru afișarea rezultatului și cele sapte butoane, șase pentru operații și un buton pentru resetarea câmpurilor.

Metodele clasei:
-getteri pentru cele două polinoame și un setter pentru rezultat, metode ce vor fi apelate în controller

-metode pentru a adăuga un ActionListener specific pentru fiecare buton din interfața (adaugaAdunaListener, adaugaScadeListener etc.)



Prezentare interfața aplicație:

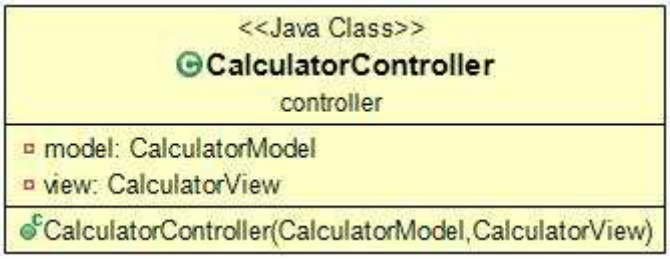


Dacă dorim să efectuăm operațiile +, -, *, / vom completa în dreptul etichetei PRIMUL POLINOM și AL DOILEA POLIMOM și vom apăsa butonul corespunzător.
Dacă dorim să efectuăm operațiile / sau integrare vom completa în dreprtul etichetei PRIMUL POLINOM și vom apăsa butonul corespunzător.
Rezultatul va fi afișat în dreptul etichetei REZULTAT.
Dacă dorim să resetam calculator vom apăsa pe butonul RESET.

4.3 Clasa din pachetul controller

- Clasa CalculatorController

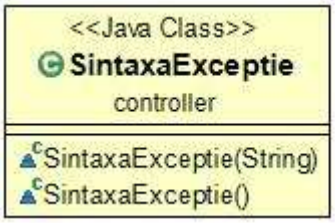
Controlorul conversează cu Modelul şi Vizualizarea.
Câmpurile clasei: un CalculatorModel și un CalculatorView.
În constructor inițializăm cele doua variabile instanță și apelăm pentru view metodele de adăugare a asculatătorilor.
În acestă clasă avem definite, de asemenea, clase membru care implementează interfața ActionListener, specifice pentru fiecare buton. De exemplu: AdunaListener, ScadeListener etc.

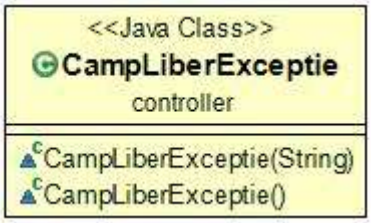


Pentru fiecare clasă implementăm metoda actionPerformed() (deoarece clasele implementează interfața ActionListener), Metoda primește un obiect de tipul ActionEvent. Spre exemplu, dacă cineva dă click pe un buton, se va recepționa un ActionEvent și se va executa ce se află în acestă metodă.

Metodele sunt implementate aproximativ la fel, deci vom descrie implementarea doar pentru una dintre acestea, pentru clasa AdunaListener. Obținem mai întai de la view ce a introdus utilizatorul în cele două casuțe de text și verificăm dacă ambele au fost completate. Dacă nu au fost, se va genera o excepție de tipul CampLiberExceptie care va fi rezolvată prin afisarea pe ecran a unui mesaj ce va semnala utilizatorului greșeala. Dacă în ambele câmpuri s-a scris ceva se trece la urmatorul pas de verificare a sintaxei. Înercăm să creeăm un polinom valid din fiecare String dat de utilizartor, dacă funcția returnează null pentru cel puțin unul din polinoame, atunci se aruncă o excepție de tipul SintaxaExceptie, care se rezolvă din nou printr-un mesaj de eroare. Dacă parsarea a avut succes, atunci se apelează funcția de inițializare a modelului cu polinoamele recent create și codul potivit operației corespunzătoare butonului apelat. La final cerem de la model rezultatul și îl afișam, prin intermediul vizualizării, utilizatorului.

Excepțiile menționate mai sus sunt, de asemenea, clase membre în clasa CalculatorController și au urmatoarele diagrame de clase.





5. Rezultate

Pentru a ne asigura că programul generează rezultatul corect vom efectua niște teste. Testarea se va realiza cu framework-ul JUnit. Vom testa doar cele șase operații care lucrează cu polinoame și implicit metoda de creeare a polinomului, însă în această etapă nu vom introduce greșeli de sinatxa, interesul fiind doar funcționarea corectă a metodelor din clasa Operatie. Pentru testarea fiecarei metode vom creea polinoamele de care avem nevoie și vom compara rezultatul (transformat în String cu metoda toString) cu Stringul pe care dorim să-l afișăm utilizatorului. Dacă acestea sunt egale în conținut atunci testarea s-a facut cu succes.

Exemple de teste pentru adunare:

P1(S)	P2(S)	REZULTAT OBTINUT(S)	REZULTAT DORIT(S)
"1-2*x"	"x"	"1-x"	"1-x"
"1-2*x"	"-1+2x"	"0"	"0"
"1-2*x"	"x^10000000"	"1-2x+x^10000000"	"1-2x+x^10000000"
"1-2*x"	"0"	"1-2x"	"1-2x"
"2*x^7+45*x^5-1"	"x^8-3x^5-5"	"-6+42x^5+2x^7+x^8"	"-6+42x^5+2x^7+x^8"
"1 * x^0 + 23x"	"2*x"	"1+25x"	"1+25x"

Exemple de teste pentru scădere:

P1(S)	P2(S)	REZULTAT OBTINUT(S)	REZULTAT DORIT(S)
"0"	"0"	"0"	"0"
"0"	"2x^3"	"-2x^3"	"-2x^3"
"-x+25x^2"	"-x+2x^3"	"25x^2-2x^3"	"25x^2-2x^3"

Exemple de teste pentru înmulțire:

P1(S)	P2(S)	REZULTAT OBTINUT(S)	REZULTAT DORIT(S)
"0"	"2x^3"	"0"	"0"
"0"	"0"	"0"	"0"
"1+x"	"1+x"	"1+2x+x^2"	"1+2x+x^2"
"x^10000"	"1+x"	"x^10000+x^10001"	"x^10000+x^10001"
"1-x+2x^3"	"-1+x-2x^3"	"-1+2x-x^2-4x^3+4x^4-4x^6"	"-1+2x-x^2-4x^3+4x^4-4x^6"

Exemple de teste pentru împarțire:

P1(S)	P2(S)		REZULTAT OBTINUT(S)	REZULTAT DORIT(S)
"0"	"2x^3"	cat:	"0"	"0"
		rest:	"0"	"0"
"1-x+2x^2"	"1+x"	cat:	"-3+2x"	"-3+2x"
		rest:	"4"	"4"
"4+3x^2+x^6"	"3x^4"	cat:	"0,3x^2"	"0,3x^2"
		rest:	"4+3x^2	"4+3x^2

Exemple de teste pentru derivare :

P1(S)	REZULTAT OBTINUT(S)	REZULTAT DORIT(S)
"0"	"0"	"0"
"1-x+2x^2"	"-1+4x"	"-1+4x"
"1-x+2x^10000+x^10"	"-1+10x^9+20000x^9999"	"-1+10x^9+20000x^9999"

Exemple de teste pentru integrare :

P1(S)	REZULTAT OBTINUT(S)	REZULTAT DORIT(S)
"0"	"0"	"0"
"1+x+2x^3"	"x+0,5x^2+0,5x^4"	"x+0,5x^2+0,5x^4"

6. Concluzii

Prin implementarea acestei teme mi-am aprofundat cunostințele de programare in Java, cunostințele de matematică leagte de polinoame (în special împarțirea polinoamelor) și am învațat cum să folosesc expresii regulate. De asemenea, am exersat și construirea unei interfețe fară ajutorul WindowBuilder, aprofunând astfel utilizarea elementelor din Java Swing.

Ca și posibilitați de dezvoltarea ulterioară, se poate lucra la interfața, o realizare mai „user friendly” este de dorit. De asemenea, mesajele de eroare la introducerea datele ar putea fi mult mai explicite. La nivel de proiectare se pot aduce înbunătațiri, iar codul poate fi optimizat.

7.Bibliografie

[1] Cursurile de tehnici de progarmare

[2] Cursurile de POO

[3] Wikipedia

[4] https://stackoverflow.com

[5] https://regex101.com/r/nI5oA5/6