Tehnici de programare fundamentale – Tema 1

Calculator de polinoame

1. Obiectivul temei

Principalul obiectiv al acestei teme este de a proiecta și implementa un calculator pentru operații cu polinoame (adunare, scădere, înmulțire, împărțire, integrare, derivare), cu o interfață grafică dedicată, concepută într-o manieră intuitivă și atractivă pentru utilizator. Proiectul își propune să îi permită utilizatorului să introducă două polinoame (sau doar unul singur, unde este cazul), să selecteze operația matematică dorită și să vizualizeze rezultatul.

Obiectivele secundare pe care dezvoltatorul și le propune se regăsesc în tabelul de mai jos, alături de secțiunile din prezenta lucrare în care vor fi adresate:

	Obiectiv secundar	Scurtă descriere	Secțiunea dedicată
1.	Analiza problemei	- stabilirea cerințelor la care programul trebuie să răspundă, și clarificarea fundamentelor teoretice implicate	2. Analiza problemei, modelare, scenarii, cazuri de utilizare
2.	Modelare	- imaginarea modelului conceptual pentru programul ce urmează a fi conceput	2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3.	Scenarii posibile	- analiza exhaustivă a situațiilor în care programul poate fi pus de către utilizator (input valid / input invalid, număr incorect de polinoame, excepție matematică)	2. Analiza problemei, modelare, scenarii, cazuri de utilizare
4.	Cazuri de utilizare	- determinarea cazurilor pentru care utilizatorul dorește să folosească programul creat	2. Analiza problemei, modelare, scenarii, cazuri de utilizare
5.	Proiectare	- definirea structurilor de date necesare, a claselor și interfețelor ce urmează a fi implementate	3. Proiectare
6.	Implementare	- descrierea claselor și a metodelor definite, explicarea funcționării și a utilității acestora; descrierea interfeței grafice	4. Implementare
7.	Testare	- testarea cu JUnit5 pentru asigurarea conformității cu rezultatele așteptate	5. Rezultate

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

2.1. Analiza problemei

S-au stabilit cerințele funcționale și non-funcționale pe care programul, calculatorul de polinoame, trebuie să le soluționeze:

a) Cerințe funcționale:
   a) calculatorul trebuie să permită utilizatorului să introducă unul sau două polinoame [1]
   b) calculatorul trebuie să permită utilizatorului să selecteze operația matematică dorită
   c) calculatorul trebuie să permită utilizatorului să vizualizeze rezultatul calculului [2]
   d) calculatorul trebuie să permită inițierea facilă a unui potențial nou calcul imediat după finalizarea calculului anterior [3]
   e) calculatorul trebuie să permită cât mai multe forme de introducere a inputului [4]
   f) calculatorul trebuie să informeze utilizatorul dacă inputul său nu este valid
   g) calculatorul trebuie să afișeze outputul respectând același șablon la fiecare utilizare[5]
   h) calculatorul trebuie să adune două polinoame
   i) calculatorul trebuie să scadă două polinoame
   j) calculatorul trebuie să înmulțească două polinoame
   k) calculatorul trebuie să împartă două polinoame [6]
   l) calculatorul trebuie să integreze un polinom [7]
   m) calculatorul trebuie să deriveze un polinom

b) Cerințe non-funcționale:
   - calculatorul trebuie să fie ușor de folosit și intuitiv
   - calculatorul trebuie să aibă o interfață atractivă

Note:

[1] – În cazul în care operația are nevoie de un singur operand, primul polinom va fi cel luat în considerare pentru calcul. Câmpurile necompletate pentru polinoamele input se vor echivala cu polinomul nul (P(x) = 0).

[2] – Pentru operația de împărțire, câmpul dedicat rezultatului va conține două polinoame, cu indicarea lor explicită prin cuvintele cheie "Quotient" – pentru cât, respectiv "Remainder" – pentru rest.

[3] – S-a decis includerea unui buton de *reset* prin acționarea căruia calculatorul este adus în starea inițială (câmpurile de input și output necompletate)

[4] – Utilizatorul trebuie să poată avea cât mai multe variante în care să introducă inputul: fără spații, cu unul sau mai multe spații între oricare două caractere; 1x, x sau x^1 să fie valide; -1x, -x sau -x^1 să fie valide; 2 sau 2x^0 să fie ambele valide; 3x^5, 3 x ^ 5, 3x ^5 să fie toate valide etc. (mai multe în secțiunea *4. Implementare,* explicarea clasei *PatternMatching*)

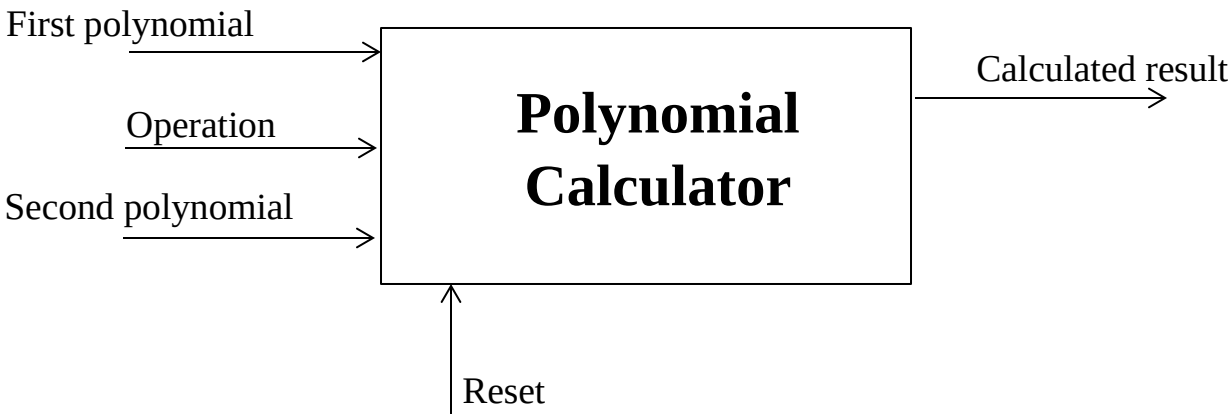
[5] – Utilizatorul trebuie să vizualizeze rezultatele într-o formă uzuală: puterile polinomului ordonate descrescător; doar termeni cu coeficienți nenuli; termenii liberi afișați strict numeric (ex: se va afișa 7, nu 7x^0); termenii de grad 1 vor fi afișați fără puterea 1 a lui x (ex: se va afișa 7x, nu 7x^1); coeficienții întregi vor fi afișați ca întregi (fără virgulă), iar cei flotanți cu un număr maxim de zecimale ce să asigure acuratețea, dar să nu încarce prea mult vizualizarea (s-a ales ca numerele în virgulă mobilă să fie afișate cu cel mult 4 zecimale).

[6] – Dacă se va încerca împărțirea cu polinomul nul, rezultatul va fi "Quotient: 0 Remainder: 0"

[7] – Constanta C rezultată la integrare se va considera 0

2.2. Modelare

S-a definit modelul de lucru al calculatorului de polinoame, sub formă de schemă bloc:



2.3. Scenarii posibile

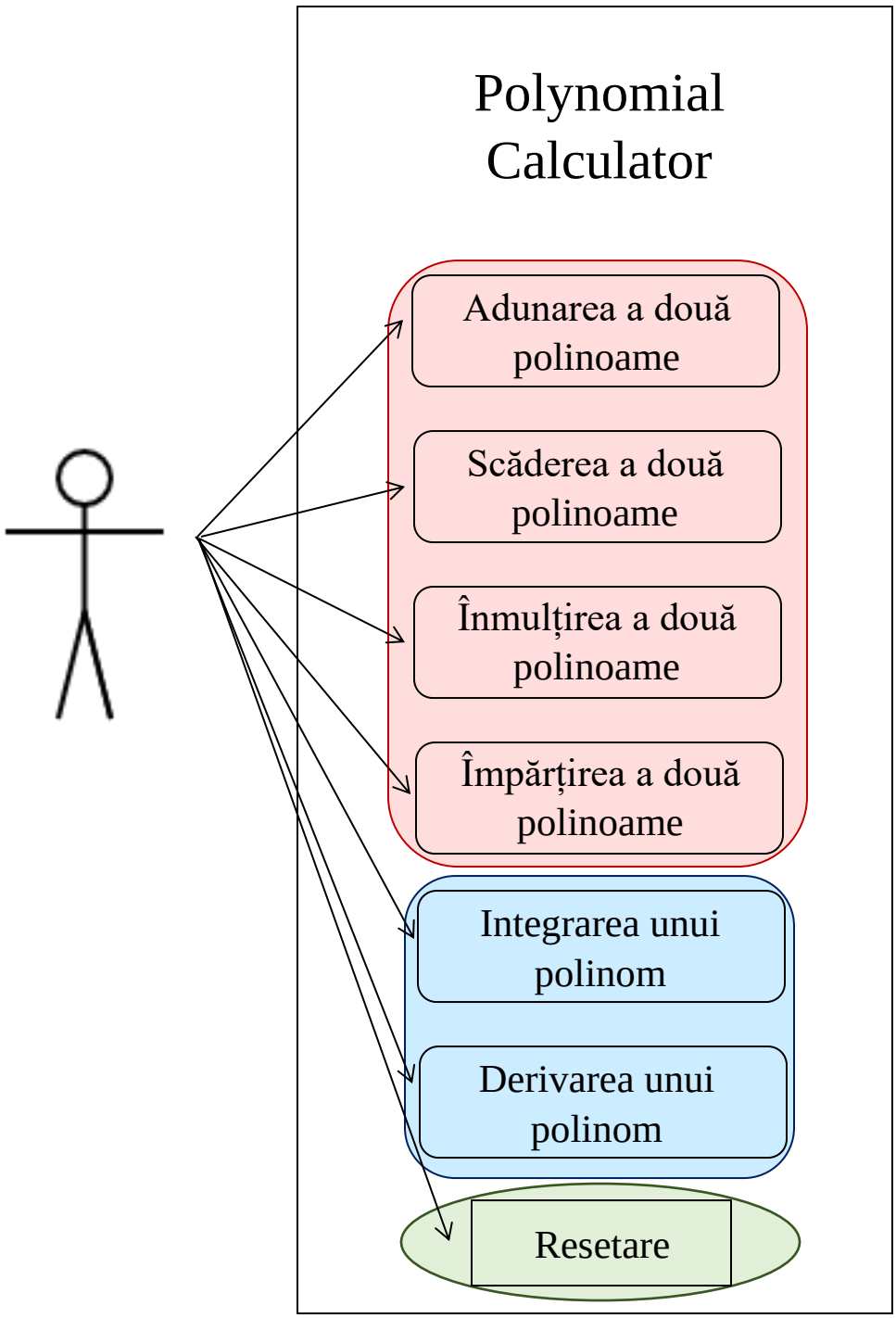
Scenariile în care se poate găsi calculatorul de polinoame pot fi diferențiate în două mari categorii: scenarii uzuale de utilizare și scenarii limită, cu subcategoriile aferente. În cele ce urmează se va prezenta felul în care programul va răspunde la fiecare dintre aceste tipologii de scenarii:

I. Scenariu uzual:
- utilizatorul introduce unul sau două polinoame valide
- utilizatorul alege operația dorită
- programul calculează rezultatul și apoi îl afișează

II. Scenarii limită:
    a. Polinom invalid:
        - utilizatorul introduce unul sau două polinoame care prezintă cel puțin o neconcordanță cu tiparul ales pentru polinom (folosește altă variabilă decât x sau X; folosește alte caractere decât cifrele 0-9, +, -, ^, spațiu, și, desigur x/X; folosește de mai multe caractere +, -, ^ una după alta etc.)
        - utilizatorul alege operația ce dorește a fi executată
        - programul îl notifică despre faptul că nu a furnizat un input valid, și îi dă posibilitatea de a corecta eroarea => utilizatorul are posibilitatea să intre într-un scenariu uzual
    b. Număr incorect de polinoame:
        - Un singur polinom introdus, în loc de două:
            - utilizatorul introduce un singur polinom valid; un câmp pentru input rămâne necompletat
            - utilizatorul alege o operație cu doi operanzi (adunare, scădere, înmulțire, împărțire)
            - programul consideră implicit că acel câmp rămas necompletat este echivalent cu polinomul nul, realizează calculul cerut și afișează rezultatul
        - Două polinoame introduse, în loc de unul:
            - utilizatorul introduce două polinoame valide
            - utilizatorul alege o operație cu un singur operand (integrare, derivare)
            - programul alege primul polinom pentru calcul și îl ignoră pe al doilea, face calculul corespunzător și afișează rezultatul
        - Niciun polinom introdus:
            - utilizatorul nu introduce niciun polinom
            - utilizatorul alege operația dorită (împărțirea este tratată și în scenariul imediat următor, *Excepție matematică*)
            - programul consideră ambele polinoame ca fiind polinomul nul, face calculele în mod normal și afișează rezultatul
    c. Excepție matematică:
        - utilizatorul introduce un polinom valid pentru primul polinom (sau nu introduce nimic și se consideră polinomul 0)
        - utilizatorul lasă câmpul pentru al doilea polinom necompletat, sau introduce polinomul nul
        - utilizatorul selectează operația de împărțire
        - programul evită intrarea într-un caz de excepție matematică și afișează ca rezultat "Quotient: 0 Remainder: 0"

2.4. Cazuri de utilizare

Se vor prezenta aici opțiunile pentru care utilizatorul le are atunci când dorește să folosească acest calculator de polinoame:



Descrierea celor 3 tipuri de cazuri de utilizare:

i. Operație cu doi operanzi:
   - utilizatorul introduce de la tastatură două polinoame valide în câmpurile marcate "First polynomial", respectiv "Second polynomial"
   - utilizatorul selectează operația dorită prin click pe butonul corespunzător din interfața grafică:
     - **+** pentru adunare
     - **-** pentru scădere
     - **x** pentru înmulțire
     - **/** pentru împărțire
   - programul calculează rezultatul și îl afișează în câmpul marcat "Calculated result"

ii. Operație cu un singur operand:
   - utilizatorul introduce de la tastatură polinomul valid în câmpul marcat "First polynomial"
   - utilizatorul selectează operația dorită prin click pe butonul corespunzător din interfața grafică:
     - ∫ pentru integrare
     - **d/dx** pentru derivare
   - programul calculează rezultatul și îl afișează în câmpul marcat "Calculated result"

iii. Resetare
   - Utilizatorul acționează prin click butonul marcat "Reset" din interfața grafică
   - Câmpurile "First polynomial", "Second polynomial" și "Calculated result" devin necompletate, indiferent de conținutul lor anterior acțiunii de resetare

3. Proiectare

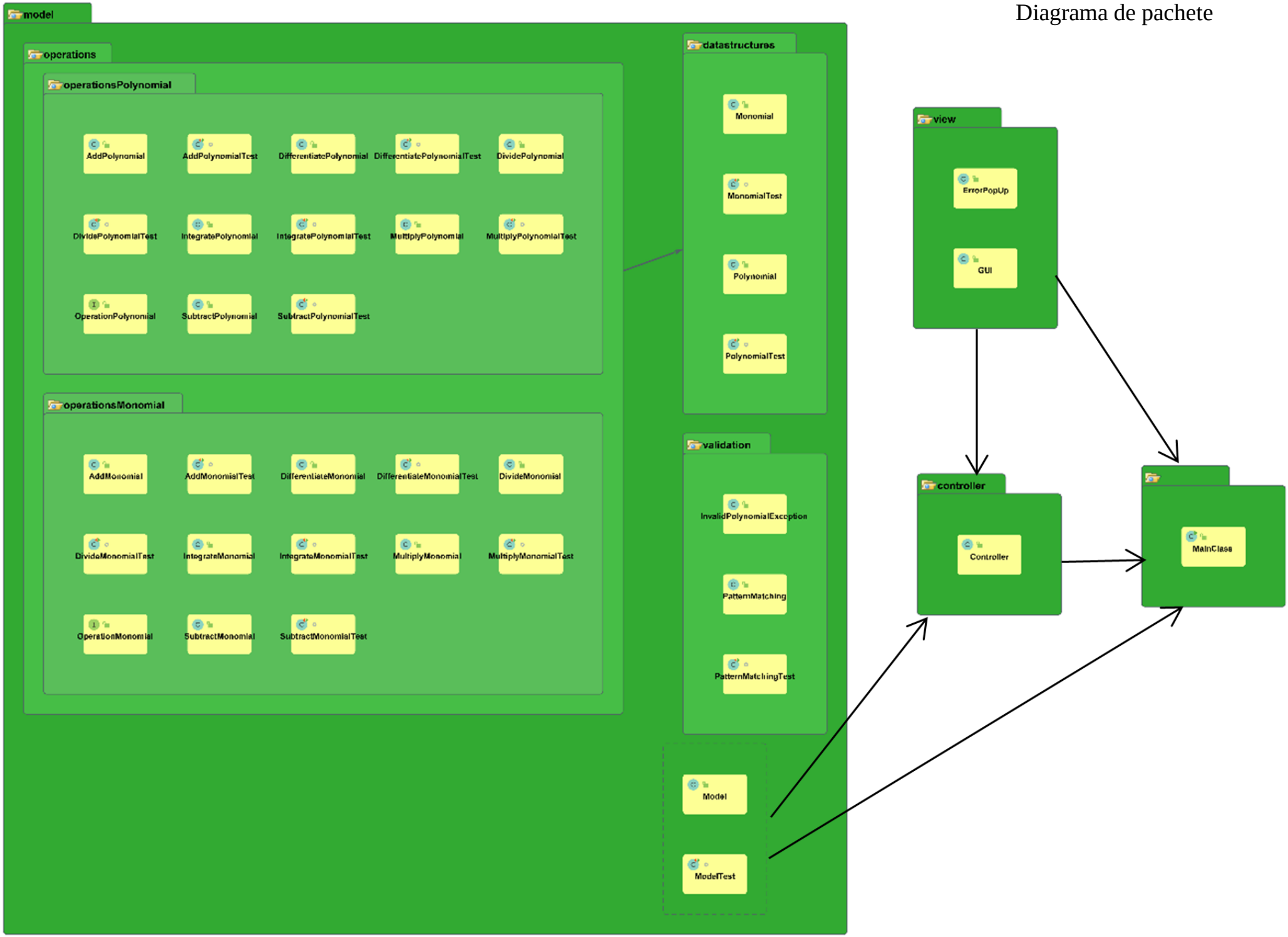
Pentru a respecta conceptele programării orientate pe obiect, fiecare clasă implementată este cât mai specifică, având cât mai puține funcționalități (ex: clasă separată pentru fiecare operație pe monom, respectiv pe polinom).

3.1. Pachete

Tiparul arhitectural folosit este MVC (Model – View - Controller), ceea ce înseamnă că partea de interfață este definită independent de restul proiectului, în pachetul View, partea de logică și calcul matematic, la fel, este de-sine-stătătoare, și se regăsește în pachetul Model. Pachetul Controller, prin clasa cu același nume, face legătura între Model și View, definind felul în care programul trebuie să se comporte în funcție de inputul utilizatorului.

Proiectul este structurat pe unități funcționale cât mai specifice, clasele care se ocupă de o anumită ramură a programului fiind grupate în pachete și sub-pachete, după cum se poate vedea mai jos, în diagrama de pachete, și în lista claselor așa cum apare ea în IDE-ul IntelliJ:

Diagrama de pachete



3.2. Clase

Diagrama de clase a întregului proiect arată relațiile dintre cele 30 de clase și interfețe definite, și este atașată pe următoarea pagină. Clasele vor fi descrise detaliat în secțiunea 4.Implementare.

3.3.Interfețe definite

Interfețele definite în proiect sunt *OperationMonomial* și *Operation Polynomial*, fiecare dintre acestea având o singură metodă abstractă, *calculate.*

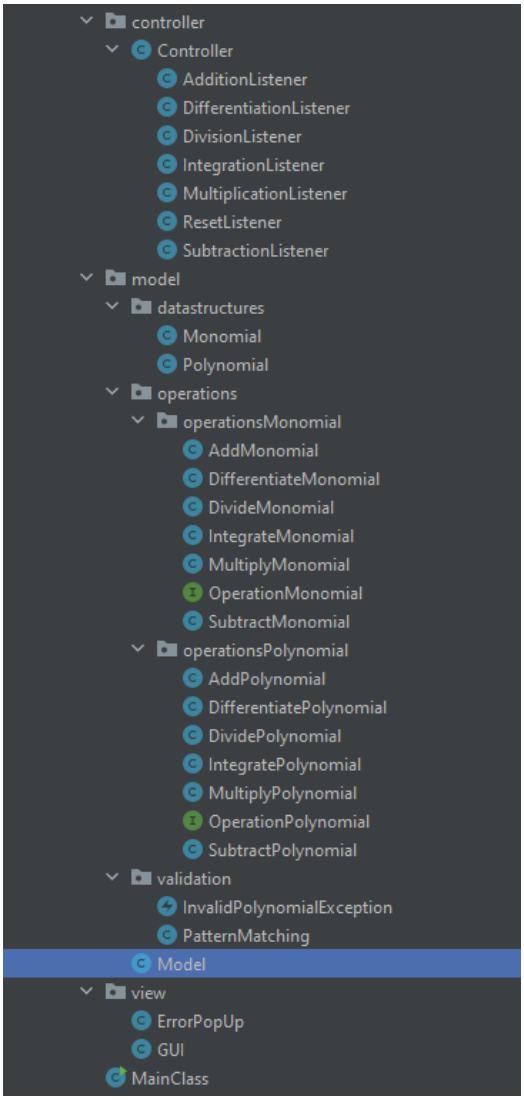
În *OperationMonomial*, *calculate* returnează o variabilă de tip *Monomial*, și are ca parametri două variabile de tip *Monomia*l. Clasele ce implementează această interfață sunt:

- *AddMonomial*
- *SubtractMonomial*
- *MultiplyMonomial*
- *DivideMonomial*
- *IntegrateMonomial*
- *DifferentiateMonomial*

Fiecare dintre acestea oferă o implementare nouă pentru metoda *calculate*, în conformitate cu operația pe care o descrie.

În *OperationPolynomial*, calculate returnează o listă de variabile de tip *Polynomial* (pentru cazul special al operației de împărțire, unde se dorește reținerea câtului și a restului, așadar două polinoame în loc de unul). Parametri metodei calculate din *OperationPolynomial* sunt doua variabile de tip *Polynomial*. Clasele ce implementează această interfață sunt:

- *AddPolynomial*
- *SubtractPolynomial*
- *MultiplyPolynomial*
- *DividePolynomial*
- *IntegratePolynomial*
- *DifferentiatePolynomial*



Structura proiectului.
Organizarea pe pachete

Fiecare clasă oferă o implementare particulară a metodei din interfață, în concordanță cu operația denumită de clasă.

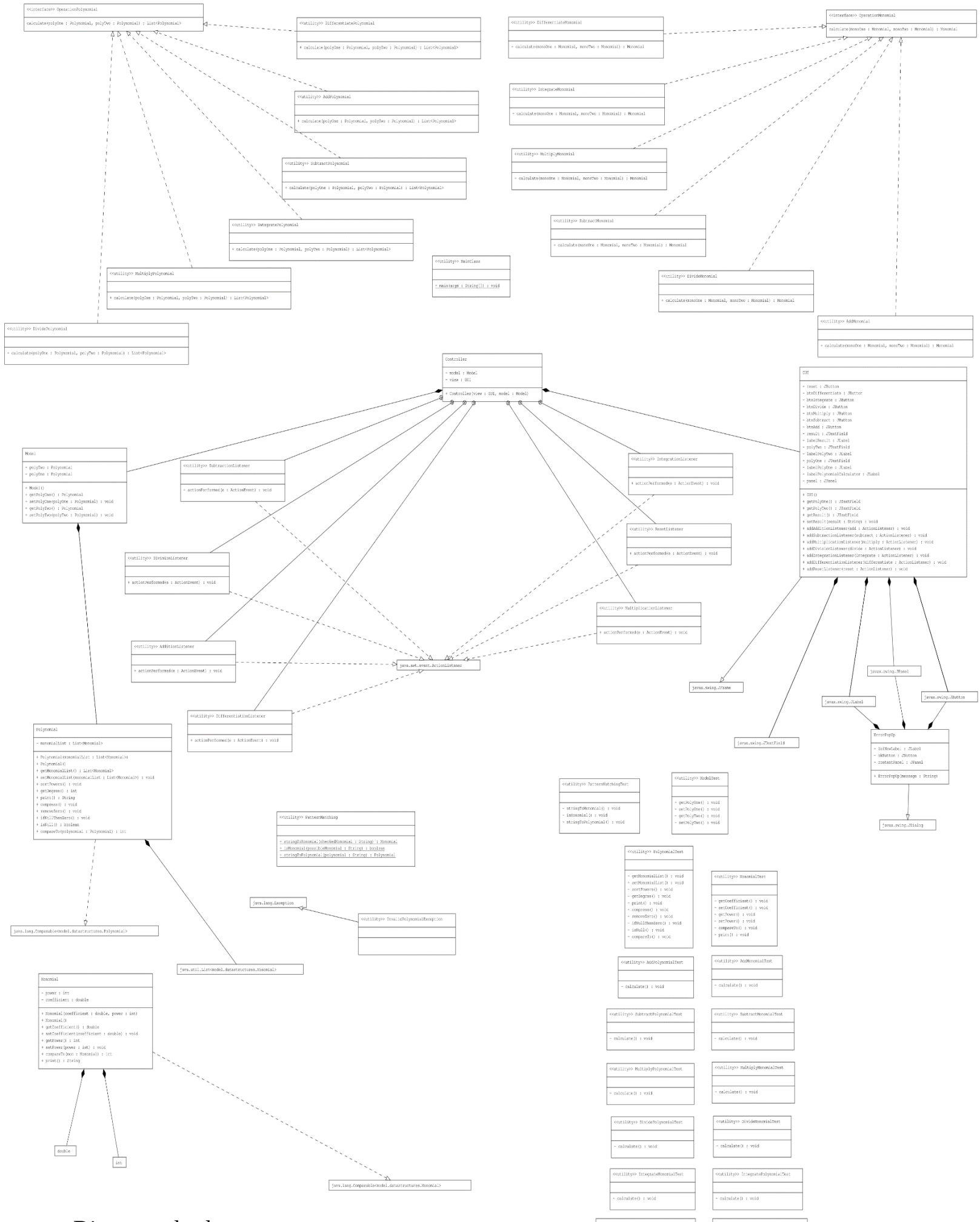


Diagrama de clase

3.4. Structuri de date folosite

Din categoria structurilor de date s-au folosit *List* și *ArrayList*. Spre exemplu, un polinom este reținut ca o listă de monoame. După cum s-a detaliat anterior, rezultatul metodei calculate implementată pe polinoame este tot o listă, însă o listă de polinoame de această dată.

Clasele *Monomial* și *Polynomial* pot fi considerate și ele structuri de date, întrucât acționează ca un tot-unitar în cadrul programului de față.

3.5. Algoritmi folosiți

Pentru împărțirea a două polinoame s-a folosit algoritmul *Polynomial long division*, care este de fapt împărțirea clasică de polinoame:

- Se ordonează monoamele ce compun fiecare cele două polinoame în ordinea descrescătoare a gradelor
- Primul monom al primului polinom se împarte la primul monom al celui de-al doilea polinom => s-a obținut primul monom al câtului
- Se înmulțește câtul cu cel de-al doilea polinom
- Rezultatul înmulțirii se scade din primul polinom => rest intermediar
- Se repetă pașii anteriori considerând restul intermediar drept nou deîmpărțit, până când gradul restului intermediar este mai mic decât gradul celui de-al doilea polinom
- Ultimul rest intermediar este restul împărțirii

În cadrul operațiilor de adunare și de scădere a două polinoame s-a adaptat algoritmul de interclasare a două liste, pentru a nu mai fi nevoie de reordonarea monoamelor rezultatului de la gradul cel mai mare la gradul cel mai mic.

4. Implementare

S-a folosit conceptul încapsulării din paradigma programării orientate pe obiect, atributele claselor fiind declarate *private* și accesibile claselor exterioare prin metode *getter* și *setter*.

4.1. Clasele pachetului *model.datastructures*

4.1.1. *Monomial:*

Clasă care implementează interfața *Comparable <Monomial>*, pentru a putea compara două monoame după gradul lor.

Monomial are două variabile instanță: *coefficient* (de tip double, chiar dacă utilizatorul trebuie să introducă doar coeficienți întregi, deoarece prin integrare sau împărțire, coeficienții rezultatului pot fi numere zecimale), și *power* (de tip int, reprezentând puterea lui x în cadrul monomului).

Se folosesc doi constructori pentru *Monomial*: unul cu doi parametri care vor dicta valorile celor două variabile instanță, iar cel de-al doilea fără parametri, care la instanțierea unui monom va considera coeficientul și puterea nule.

S-au definit metodele mutatoare *setCoefficient* și *setPower*, și cele accesoare *getCoefficient* și *getPower*, cu rolul de a permite accesul la variabilele instanță private ale clasei.

Metoda *public int compareTo (Monomial mon)* este folosită pentru a compara puterile a două monoame. Ea returnează 1 dacă monomul curent are gradul mai mare decât monomul *mon,* -1 dacă monomul curent are gradul mai mic decât *mon*, și 0 dacă monomul curent și monomul transmis ca parametru au același grad. Este important de remarcat faptul că 0 ca rezultat returnat de metoda *compareTo* din *Monomial* nu înseamnă că monomul curent și cel parametru sunt identice; comparația se referă strict la atributul *power* al obiectelor *Monomial*. Această metodă este folosită la sortarea monoamelor în cadrul unui polinom.

Metoda *public String print()* listează ca String polinomul curent. Listarea se va face în formatul uzual de scriere matematică: dacă puterea lui x este 0, atunci se va afișa doar coeficientul lui x; dacă puterea lui x este 1, atunci se va afișa doar x, nu x^1; dacă monomul are coeficientul 1 sau -1, acest coeficient va fi ignorat la listare; dacă monomul are coeficientul număr pozitiv, atunci el va fi precedat de +, altfel va fi precedat de - ; dacă monomul are coeficientul număr întreg, atunci el se va scrie fără zecimalele .0000 , altfel coeficientul se va scrie cu maxim 4 zecimale exacte (ex: 1/3 va fi scris ca 0,3333, iar 1/2 ca 0,5).

4.1.2. *Polynomial*

Clasă care implementează interfața *Comparable <Polynomial>*, pentru a putea testa ușor identitatea a două polinoame (două polinoame sunt identice dacă au aceleași puteri ale lui x asociate acelorași coeficienți).

Singura variabilă instanță a clasei *Polynomial* este de tipul *List <Monomial>*, și ilustrează în mod natural felul în care este construit un polinom: ca o înșiruire de monoame.

Clasa are doi constructori: unul fără parametri, care doar alocă lista de monoame din cadrul polinomului, iar celălalt cu un parametru de tip *List <Monomial>,* care atribuie lista parametru listei de monoame a polinomului curent.

Metodele *setMonomialList* și *getMonomialList* au scopul de a permite accesarea listei de monoame ce compune polinomul curent.

Metoda *public void sortPowers()* realizează ordonarea descrescătoare a monoamelor din lista de monoame a polinomului, astfel încât în final polinomul să aibă structura:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \cdots + a_2 x^2 + a_1 x + a_0$$

Metoda *public int getDegree()* returnează gradul polinomului curent.

Pentru listarea polinomului s-a folosit *public String print()*, care apelează intern metoda de afișare pentru monoame. S-a ținut cont de faptul că dacă primul monom al polinomului are coeficient pozitiv, nu este de dorit să se afișeze și semnul + înaintea monomului (ex: x^2 - 2x + 1, nu + x^2 - 2x + 1).

public void compress() este metoda definită pentru a asigura că nu există mai multe monoame cu același grad în interiorul polinomului. Se poate ajunge în această situație dacă utilizatorul introduce polinomul sub această formă (ex: x^3 + x^2 + x + 1 + 2x^2 + 2 => x^3 + 3x^2 + x + 3), sau dacă la înmulțirea a două polinoame puterile lui x din rezultat pot fi obținute ca sumă a mai mult de o combinație dintre puterile polinomului unu cu puterile polinomului 2 (ex: (x^2 + x + 1) * (2x + 1) => obțin puterea 2 a lui x și înmulțind x^2 * 1, și înmulțind x * 2x).

public void removeZero() elimină monoamele cu coeficientul 0 din compoziția polinomului curent. Zero drept coeficient poate fi intodus de către utilizator sau poate rezulta în urma operațiilor de adunare, scădere, înmulțire, împărțire.

Pentru a reuși afișarea polinomului nul, și realizarea operațiilor cu acesta fără a genera o excepție de tipul *NullPointerException* s-a definit metoda *public void ifNullThenZero*. Dacă polinomul curent nu are niciun monom în lista sa de definiție, atunci se inserează un monom cu coeficientul 0 și puterea 0 prin apelul acestei metode.

Pentru a determina dacă polinomul nu are niciun monom în lista lui, se apelează metoda *public boolean isNull()*.

Metoda *public int compareTo (Polynomial polynomial)* este folosită strict pentru a determina dacă două polinoame sunt identice, și este necesară pentru partea de testare unitară a calculatorului. Metoda returnează 0 dacă polinomul curent și cel transmis ca parametru sunt identice, și -1 în rest.

4.2. Clasele pachetului *model.operations.operations_monomial*

Acesta este pachetul în care este definită interfața *OperationMonomial*, descrisă anterior în secțiunea 3.3. Toate clasele din acest pachet (sub-pachet) implementează interfața *OperationMonomial* și dau o implementare proprie metodei *public Monomial calculate(Monomial monoOne, Monomial monoTwo)*. Toate cele 6 clase au doar constructorul implicit.

4.2.1. *AddMonomial*

Clasă care implementează metoda de adunare a două monoame cu același grad. Rezultă un monom cu același grad și cu coeficientul egal cu suma coeficienților celor două monoame trimise ca parametri.

4.2.2. *SubtractMonomial*

Clasă care implementează metoda de scădere a două monoame cu același grad. Rezultă un monom cu același grad și cu coeficientul egal cu diferența coeficienților celor două monoame trimise ca parametri.

4.2.3. *MultiplyMonomial*

Clasă care implementează metoda de înmulțire a două monoame. Rezultă un monom cu gradul egal cu suma gradelor celor doua monoame trimise ca parametri, și coeficientul egal cu produsul coeficienților celor două monoame.

4.2.4. *DivideMonomial*

Clasă care implementează metoda de împărțire a două monoame. Rezultă un monom cu gradul egal cu diferența gradelor celor doua monoame trimise ca parametri, și coeficientul egal cu rezultatul real al împărțirii coeficienților celor două monoame.

4.2.5. *IntegrateMonomial*

Clasă care implementează metoda de integrare a unui singur monom. Al doilea monom trimis ca parametru este complet ignorat în cadrul metodei, și este transmis doar pentru a respecta semnătura metodei *calculate* din interfața *OperationMonomial*. Rezultă un monom cu gradul egal cu gradul monomului trimis ca prim parametru, crescut cu o unitate, și coeficientul egal cu rezultatul real al împărțirii coeficientului primului monom la 1 + gradul primului monom trimis ca parametru.

4.2.6. *DifferentiateMonomial*

Clasă care implementează metoda de derivare a unui singur monom. Al doilea monom trimis ca parametru este ignorat în cadrul metodei, și la fel ca în cazul clasei *IntegrateMonomial*, este transmis doar pentru a respecta semnătura metodei *calculate* din interfața implementată. Rezultă un monom cu gradul egal cu gradul monomului trimis ca prim parametru, scăzut cu o unitate, și coeficientul egal cu rezultatul real al înmulțirii dintre coeficientul și gradul primului monom. Descrierea este valabilă în cazul monoamelor având gradul cel puțin egal cu 1. Monoamele de grad 0 sunt constante față de x la derivare, deci devin 0.

4.3. Clasele pachetului *model.operations.operations_polynomial*

În acest pachet este definită interfața OperationPolynomial, prezentată detaliat în secțiunea 3.3. Toate clasele din pachetul model.operations.operations_polynomial implementează această interfață, implementând implicit și metoda *public List<Polynomial> calculate(Polynomial polyOne, Polynomial polyTwo)*. Cele 6 clase din pachetul de față au doar constructorul default.

4.3.1. *AddPolynomial*

Clasă care implementează metoda de adunare a două polinoame. Se ordonează monoamele celor două polinoame în ordine descrescătoare a gradelor, apoi se calculează suma dintre monoamele din primul polinom și cele din al doilea polinom care au același grad, prin parcurgerea cu două cursoare ce se deplasează gradual și interacționează similar unei interclasări. Rezultă polinomul sumă cu lista monoamelor deja ordonată. Se elimină din rezultat monoamele care au coeficientul 0, iar dacă lista monoamelor este nulă se adaugă un monom cu coeficientul 0 si puterea 0. Rezultatul este transmis ca prim și unic element al listei de polinoame returnate.

4.3.2. *SubtractPolynomial*

Clasă care implementează metoda de scădere a două polinoame. Se ordonează monoamele celor două polinoame în ordine descrescătoare a gradelor. Se calculează diferența dintre monoamele din primul polinom și cele din al doilea polinom care au același grad, cu ajutorul metodei de calcul a diferenței de monoame, asemeni unei parcurgeri de interclasare. Rezultă polinomul diferență cu lista monoamelor deja ordonată. Se elimină din rezultat monoamele care au coeficientul 0. Dacă lista monoamelor este nulă se adaugă un monom cu coeficientul 0 si puterea 0, pentru a putea realiza fără probleme operațiile conexe sau afișarea. Rezultatul este transmis ca singurul element al listei de polinoame returnate.

4.3.3. *MultiplyPolynomial*

Clasă care implementează metoda de înmulțire a două polinoame. Se înmulțește fiecare monom al primului polinom cu fiecare monom al celui de-al doilea polinom. Dacă vreun monom obținut are coeficientul 0, monomul respectiv se ignoră. Se apelează *result.compress()* pentru a aduna monoamele cu același grad din rezultat. Dacă rezultatul este un polinom cu lista monoamelor nulă, atunci se adaugă în lista lui de monoame un monom cu coeficientul 0 și puterea 0. Prin apelul *result.sortPowers()* se sortează în ordine descrescătoare monoamele ce compun polinomul. Polinomul rezultat se adaugă la lista de polinoame ce va fi returnată, iar apoi se returnează lista cu acest singur polinom.

4.3.4. *DividePolynomial*

Clasă care implementează metoda de împărțire a două polinoame. Se ordonează monoamele celor două polinoame în ordine descrescătoare a gradelor. Se elimină monoamele cu coeficient nul din cele două polinoame. Se aplică algoritmul de împărțire a două polinoame – *Polynomial long division* – explicat pe larg în secțiunea 3.5. Rezultă polinoamele cât și rest, pentru care se aplică preventiv *compress()*, se rezolvă problema de afișare în cazul polinomului nul, și se adaugă cele două polinoame la lista de polinoame returnată, câtul la indexul 0, iar restul la indexul 1.

4.3.5. *IntegratePolynomial*

Clasă care implementează metoda de integrare a unui singur polinom. Cel de-al doilea polinom trimis ca parametru este neglijabil. Se parcurge lista de monoame a primului polinom și se aplică metoda *calculate()* din clasa *IntegrateMonomial*, pentru a găsi lista de monoame a polinomului rezultat. Se asigură forma potrivită pentru viitoarea afișare a rezultatului prin apelurile *result.compress(), result.sortPowers(), result.ifNullThenZero()*. Se adaugă rezultatul la lista de polinoame ce va fi returnată.

4.3.6. *DifferentiatePolynomial*

Clasă care implementează metoda de derivare a unui polinom. Al doilea polinom trimis ca parametru este ignorat. Se parcurge lista de monoame a primului polinom și se aplică metoda *calculate()* din clasa *DifferentiateMonomial*, pentru a găsi lista de monoame a polinomului rezultat, care este apoi adus la forma pentru afișare prin apelurile *result.compress(), result.sortPowers(), result.ifNullThenZero()*. Se adaugă rezultatul la lista de polinoame *list* și se returnează *list*.

4.4. Clasele pachetului *model.validation*

4.4.1. *PatternMatching*

Clasă care implementează 3 metode statice având ca scop verificarea validității inputului dat de către utilizator, și transformarea inputului (primit ca *String*) într-un obiect de tip *Polynomial.*

Metoda *public static boolean isMonomial (String possibleMonomial)* returnează *true* dacă șirul de caractere trimis ca parametru respectă tiparul unui monom, și *false* în caz contrar. Pentru această verificare s-au folosit 3 pattern-uri *regex* (*Regular Expression*): unul pentru cazul general de scriere a unui monom (ex: -7x^30; x^2; 3x^1; 4x^0), unul pentru cazul în care monomul are gradul 1 și se omite scrierea puterii lui x (ex: 15x), iar cel de-al treilea pentru monoame de gradul 0 scrise strict ca valoare numerică (12, nu 12x^0). Se permite orice număr de spații între oricare două caractere ce contribuie activ la scrierea polinomului (cifrele 0-9, x sau X, +, -, ^).

Metoda public static *Monomial stringToMonomial (String checkedMonomial)* returnează un obiect de tip *Monomial* care corespunde șirului de caractere dat ca parametru, șir deja verificat ca fiind un monom valid.

Metoda *public static Polynomial stringToPolynomial(String polynomial) throws InvalidPolynomialException* transformă șirul de caractere primit ca parametru într-un obiect de tip Polynomial dacă șirul de caractere este o enumerare validă de monoame. În caz contrar, metoda aruncă o excepție definită la nivelul programului, InvalidPolynomialException. Această excepție va fi tratată la nivelul clasei *Controller* a pachetului cu același nume cu scopul de a-i transmite utilizatorului un feedback prin interfața grafică pentru a-l înștiința că unul dintre polinoamele introduse de el nu este valid.

În implementare se folosește faptul că finalul unui monom este marcat de un caracter + sau -, sau de finalul șirului.

4.4.2. *InvalidPolynomialException*

Clasă ce extinde *Exception,* folosită pentru a semnala programului introducerea unui șir de caractere ce nu respectă convenția de scriere a unui polinom. Este aruncată de metoda *public static Polynomial stringToPolynomial(String polynomial)* și este tratată în clasele interioare ale clasei *Controller*.

4.5. Clasele pachetului *model*

4.5.1. *Model*

Clasă ce schițează modelul matematic al calculatorului și reține obiectele ce vor fi folosite mai apoi în soluționarea cererii trimise de către utilizator: pentru fiecare operație care poate fi realizată de către calculator este nevoie de cel mult două polinoame. Aceste două polinoame, obiecte de tip *Polynomial*, vor fi variabilele instanță ale clasei *Model*, ambele de tip *private*.

Clasa *Model* are un constructor fără parametri, care inițializează variabilele instanță de tip *Polynomial* cu null.

Singurele metode definite în clasa Model sunt getter-ele și setter-ele pentru fiecare dintre cele două polinoame: *public Polynomial getPolyOne(), public void setPolyOne(Polynomial polyOne), public Polynomial getPolyTwo(), public void setPolyTwo(Polynomial polyTwo)*.

4.6. Clasele pachetului *view*

4.6.1. *GUI*

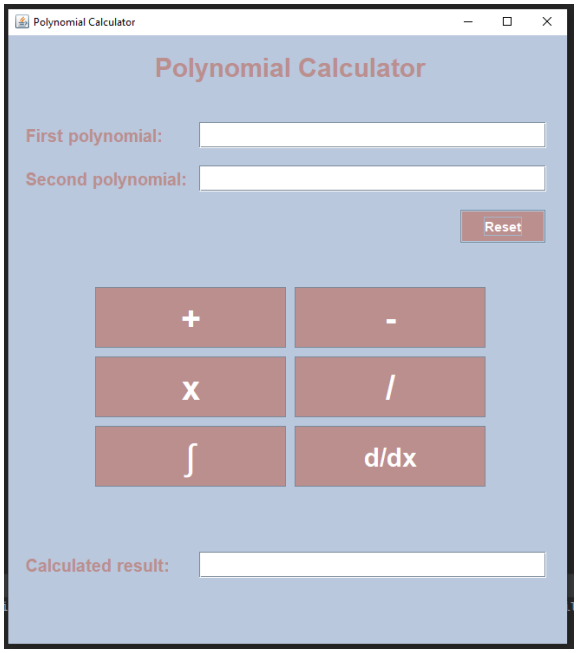
Clasa care creează interfața grafică a calculatorului. Această clasă extinde *JFrame,* ceea ce înseamnă că, în fapt, orice obiect *GUI* este o fereastră ce poate fi configurată grafic.

Toate elementele specifice Java Swing (*JPanel, JLabel, JTextField, JButton*) sunt variabile instanță de tip *private* ale obiectelor *GUI*. Se definesc doar metodele *getter* și *setter* care sunt necesare pentru transmiterea de informații de la utilizator la program și vice-versa.

Celelalte metode definite sunt cele necesare pentru adăugarea ulterioară a elementelor de interacțiune user – program în clasa *Controller*. Pentru fiecare buton s-a definit o metodă care să îi adauge un *ActionListener,* pentru a putea capta informația că un anumit buton a fost acționat.

Fereastra de interfață are titlul *Polynomial Calculator* (atât ca titlu efectiv al ferestrei, cât și ca etichetă cu rol de titlu în fereastră). Două etichete text (*JLabel*) îi indică utilizatorului care sunt câmpurile (*JTextField*) dedicate introducerii primului polinom, respectiv a celui de-al doilea.

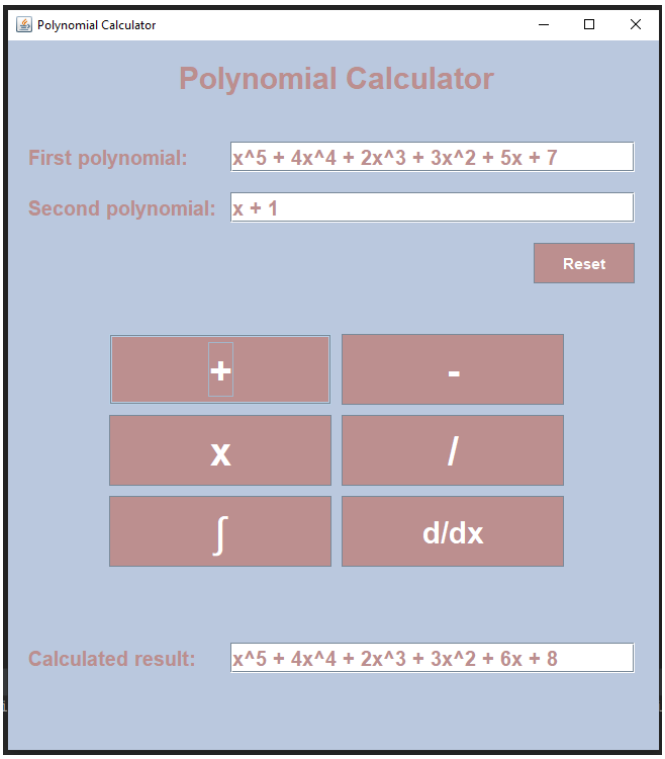
Alături de zona dedicată introducerii de polinoame se află și butonul de reset, prin acționarea căruia fereastra este adusă înapoi în stadiul inițial – toate câmpurile vide.

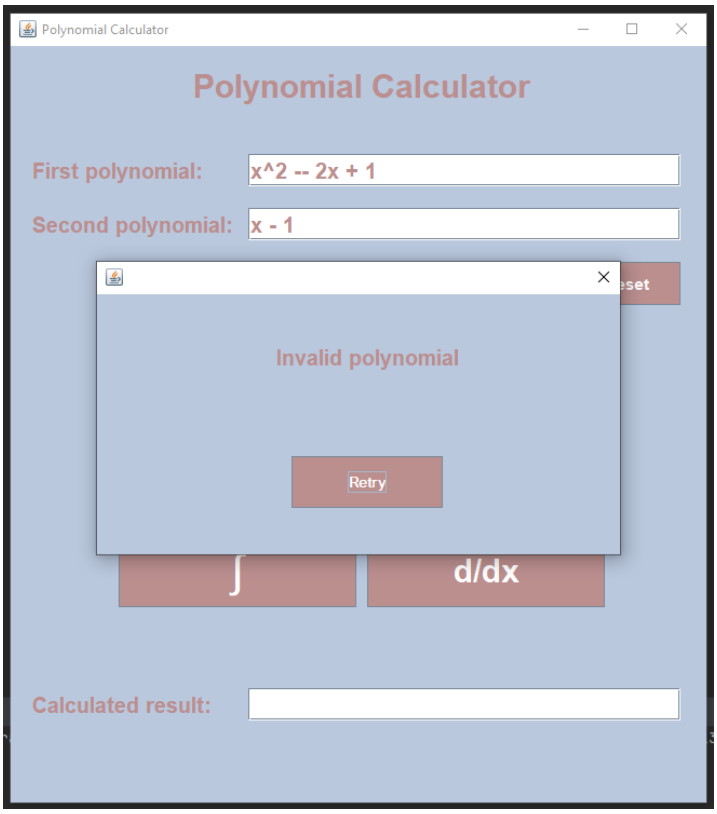


Dedesubt se află 6 butoane asociate celor 6 operații pe care calculatorul de polinoame poate să le facă, marcate după cum urmează:

- \+ pentru adunare
- \- pentru scădere
- x pentru înmulțire
- / pentru împărțire
- ∫ pentru integrare
- d/dx pentru derivare

La apăsarea oricăruia dintre aceste butoane, programul calculează rezultatul și îl afișează în câmpul de mai jos, marcat de eticheta *Calculated Result.*





4.6.2. *ErrorPopUp*

Clasă care extinde *JDialog*, reprezentând, de fapt, caseta de dialog prin care utilizatorul este avertizat că inputul său nu este valid.

Aceasta afișează un mesaj sugestiv și îi dă user-ului posibilitatea de a reveni în fereastra principală fie prin închiderea casetei de dialog, fie prin acționarea butonului *Retry* din casetă.

4.7. Clasele pachetului *controller*

4.7.1. *Controller*

Clasa *Controller* este cea care face legătura între model și view, între utilizator și calculator, între input, program și output.

Controller are ca atribute un obiect de tip *GUI* și unul de tip *Model*, și reprezintă, practic, o colecție de clase interioare care implementează interfața *ActionListener*, clase care mai apoi vor fi folosite pentru a le adăuga *listeneri* butoanelor din interfață, fiecare buton executând operația corespunzătoare.

Constructorul clasei primește ca parametri un *GUI* și un *Model*, pe care le atribuie obiectului curent. Tot constructorul este el care adaugă *listenerii*, implementați drept clase interioare, la butoanele din interfața grafică.

În fiecare clasă interioară (cu excepția clasei *ResetListener*) se încearcă extragerea celor două polinoame introduse de utilizator ca input. Acest lucru se face într-un bloc try-catch. Dacă operațiunea eșuează, se aruncă o excepție de tipul *InvalidPolynomialException* și se deschide o casetă de dialog de tipul *ErrorPopUp* cu mesajul *Invalid polynomial*. Dacă ambele șiruri de caractere introduse de către utilizator pot fi traduse ca două obiecte *Polynomial* la nivelul programului, atunci se instanțiază un obiect de tipul clasei de operație dorite (*AddPolynomial* pentru adunare, *SubtractPolynomial* pentru scădere etc.).

Clasa interioară *ResetListener* resetează calculatorul, câmpurile interfeței devenind goale.

Clasele interioare ale clasei *Controller*, explicate mai sus, sunt:

- *AdditionListener*
- *SubtractionListener*
- *MultiplicationListener*
- *DivisionListener*
- *IntegrationListener*
- *DifferentiationListener*
- *ResetListener*

4.8. *MainClass*

Clasa principală este singura care are implementată metoda *public static void main (String[] args)*, deci este singura clasă *runnable* a proiectului.

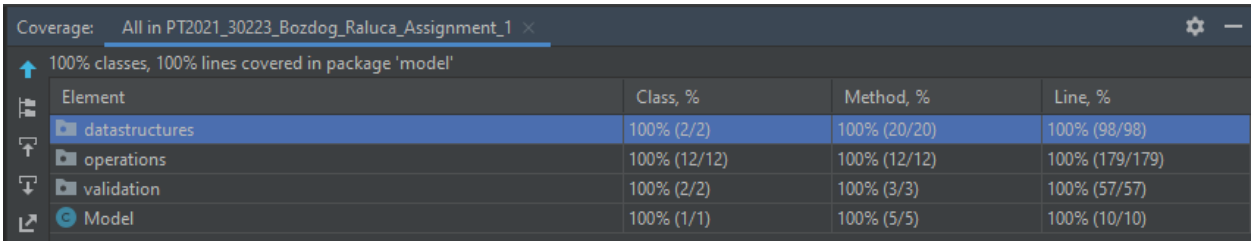
În metoda *main* a acestei clase se declară și se instanțiază un obiect de tip *Model* (*model*), unul de tip *GUI* (*view*), și unul de tip *Controller*, care are *model* și *view* ca variabile instanță. În această metodă principală este făcută vizibilă fereastra reprezentând calculatorul propriu-zis și este practic pus în funcțiune calculatorul.

5. Rezultate

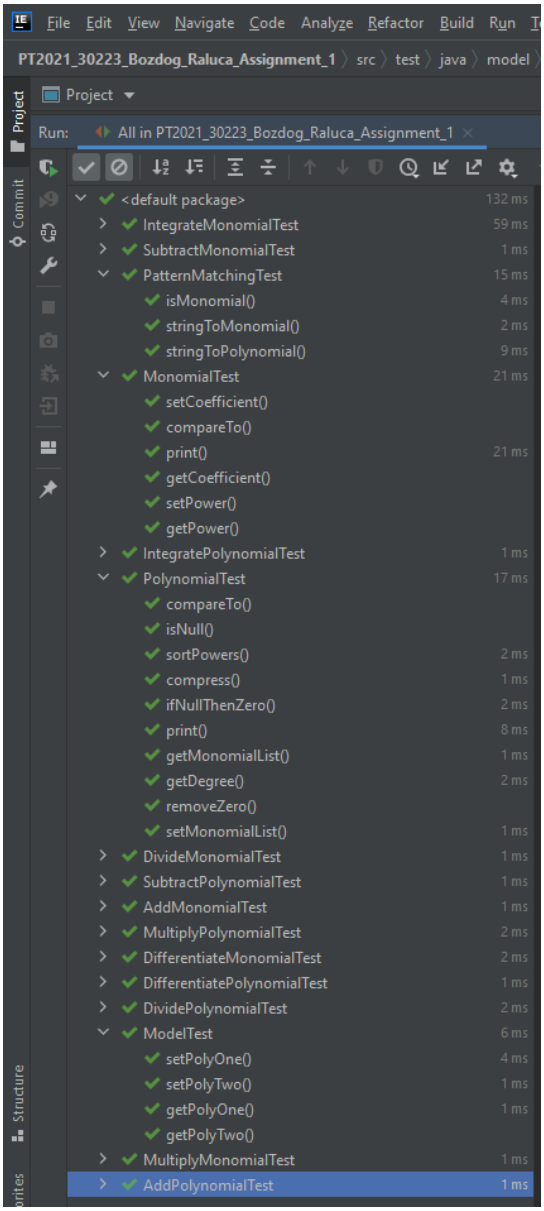
În urma definitivării proiectului, pe lângă multitudinea de teste executate doar prin rularea programului și compararea rezultatului afișat cu rezultatul calculat ca fiind matematic corect, s-a realizat și testarea cu *JUnit* a claselor din pachetul *model*, responsabil de modelarea matematică a programului.

Cel mai complex test este cel pentru clasa *PatternMatching*, care ajută la verificarea validității inputului transmis de utilizator. În clasa de test PatternMatchingTest, în metoda *stringToPolynomial()*, care în clasa originală returnează un polinom sau aruncă o excepție, s-au testat ambele scenarii posibile: polinom valid, respectiv polinom invalid. Pentru testarea scenariului de polinom neconform s-a folosit metoda *assertThrows()* pentru a testa aruncarea unei excepții *InvalidPolynomialException*.

S-au creat și completat clase, metode și situații de test până la obținerea unei acoperiri de 100% a pachetului *model*, atât în ceea ce privește clasele, cât și metodele, și chiar liniile de cod, după cum se poate vedea în imaginile din *IntelliJ* atașate mai jos:



Situația testării pachetului *model*



Toate metodele claselor din pachetul model sunt validate

Clasele din pachetul *model* sunt testate în proporție de 100%

6. Concluzii

Tema de față a prezentat elemente de noutate în primul rând prin folosirea pattern-urilor *regex,* care au constituit un element cheie în rezolvarea cerințelor.

O tehnică OOP dobândită în urma dezvoltării proiectului este folosirea modelului arhitectural MVC, prin adoptarea căruia organizarea claselor proiectului, și mai ales gestiunea legăturilor utilizator – program – utilizator au devenit mult mai clare și ușor de imaginat.

Totodată, folosirea încapsulării exclusive a datelor, prin definirea tuturor variabilelor instanță ca *private,* a fost explorată îndeaproape pentru calculatorul de polinoame.

Împărțirea claselor în pachete și sub-pachete este tot o practică însușită cu această temă.

Testarea unitară a fost aprofundată în cadrul acestui proiect, cu o acordare mai mare de atenție la detaliile cazurilor limită.

Definirea de metode cât mai puțin voluminoase a fost o adevărată provocare a creării calculatorului de polinoame, direcție în care programul ar putea să fie optimizat ulterior.

Alte direcții de dezvoltare ulterioară ar putea include:

- îmbunătățirea modului de vizualizare a rezultatului în cazul operației de împărțire (fie prin adăugarea unui câmp secund pentru rezultat strict în acest caz, fie prin transformarea *JTextField*-ului de rezultat într-o *JTextArea* care să aibă două rânduri pentru împărțire si unul singur în rest, fie prin adăugarea unui *JScrollPanel* câmpului existent pentru a permite navigarea mai facilă stânga-dreapta)
- adăugarea unor efecte 3D butoanelor ca upgrade al interfeței cu utilizatorul
- adăugarea unor butoane de cifre și simboluri (+, -, ^) care să permită user-ului să aibă încă o cale de introducere a polinoamelor input pe lângă tastatură
- modificarea calculatorului în așa fel încât la apăsarea unui buton, polinomul rezultat să fie inserat în câmpul *First polynomial*, pentru ca utilizatorul să aibă facilitatea de a realiza noi calcule cu acesta (ex: să realizeze, astfel, indirect, adunarea a 3 polinoame) direct din interfață, fără a copia conținutul lui *Calculated result* și a-l insera manual ca input
- afișarea coeficienților polinoamelor ca fracții ireductibile, și nu ca numere flotante, în cazul integrării și al împărțirii
- o mai bună comunicare cu utilizatorul prin mesaje de eroare la introducerea unui șir de caractere care nu poate fi tradus sub formă de polinom (ex: Nu puteți folosi caracterul *. Coeficienții polinomului sunt numere întregi. Polinomul trebuie să fie funcție de o singură variabilă)

7. Bibliografie

- TP2020-2021_Descriere_Laborator.pdf
- ASSIGNMENT_1_SUPPORT_PRESENTATION.pptx
- https://www.w3schools.com/java/java_regex.asp
- https://www.baeldung.com/junit-assert-exception
- https://en.wikipedia.org/wiki/Polynomial_long_division