Documentație

Tema 1: Calculator de polinoame

Moldovan Maria Alexandra
AC CTI, grupa 30224
27.03.2023

Cuprins

1. Obiective

1.1 Obiectiv principal

Proiectați un sistem de procesare a polinoamelor de o singură variabilă, implementând operațiile de adunare, scădere, înmulțire, împărțire, derivare și integrare. Polinoamele vor fi citite și reținute sub forma standard:

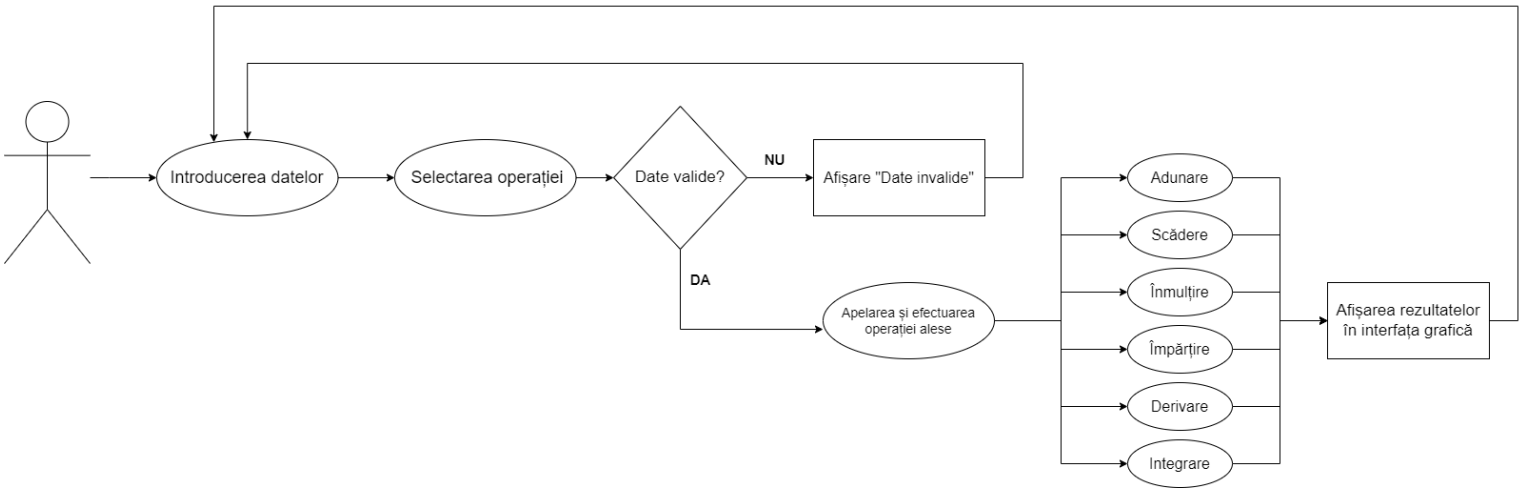
$$a\_nx\^n + a\_(n-1)x\^(n-1) + \cdots + a\_1x + a\_0,$$

unde a_i sunt coeficienții polinomului și x este variabila.

1.2 Obiective secundare

Obiectiv	Detaliere	Prezentat în capitolul
Dezvoltarea de *use cases* și scenarii	Reprezentarea diagramelor *use case*, precum și exemplificarea unor operații implementate	2
Dezvoltarea algoritmilor	Explicarea și detalierea metodelor folosite în implementarea proiectului	3
Implementarea soluției	Prezentarea funcționalităților interfeței din perspectiva utilizatorului	4
Testare	Prezentarea unor JUnitTests	5

2. Analiza problemei



Urmărind diagrama *use case* de mai sus, primul pas este introducerea de date de către utilizator, formatat de către o *Regular Expression* (Regex). În momentul apăsării butonului corespunzător operației dorite, este apelată metoda **validatePolynomial**, care preia inputul citit sub formă de String și verifică validitatea datelor. Dacă acesta nu respectă forma noastră impusă de polinom, se va afișa un mesaj corespunzător, și de asemenea, semnalează care dintre cele două polinoame a fost introdus greșit. În caz favorabil, se va trece direct la pasul de procesarea datelor.

Pentru date valide, apăsarea butonului corespunzător cu alegerea operației va transmite polinomul fomatat către metoda operației corespunzătoare, urmând ca și rezultatul acesteia să se întoarcă înapoi în interfața grafică în altă casetă, sub formă de String.

Soluția implementată de mine este una mai dificilă. Exista posibilitatea de a avea casete separate în care să se introducă pur și simplu valoarea coeficientului și a gradului pentru un monom, dar eu am ales să ofer libertate utilizatorului. Nu este nevoie să introducă monoamele din polinom în nicio ordine a gradelor, și de asemenea, poate scrie două monoame de acelasi grad în compoziția aceluiași monom, acest detaliu neaducând probleme ulterioare. De asemenea, am tratat și cazul în care unul din polinoame este 0.

3. Proiectare

3.1 Introducere

Scopul principal al acestui proiect este de a dezvolta un sistem de procesare a polinoamelor care să permită utilizatorilor să efectueze operațiile de adunare, scădere, înmulțire, împărțire, derivare și integrare a polinoamelor de o singură variabilă. Polinoamele vor fi citite și reținute sub forma standard:

$a\_nx\^n + a\_(n-1)x\^(n-1) + \cdots + a\_1x + a\_0$, unde a_i sunt coeficienții polinomului și x este variabila.

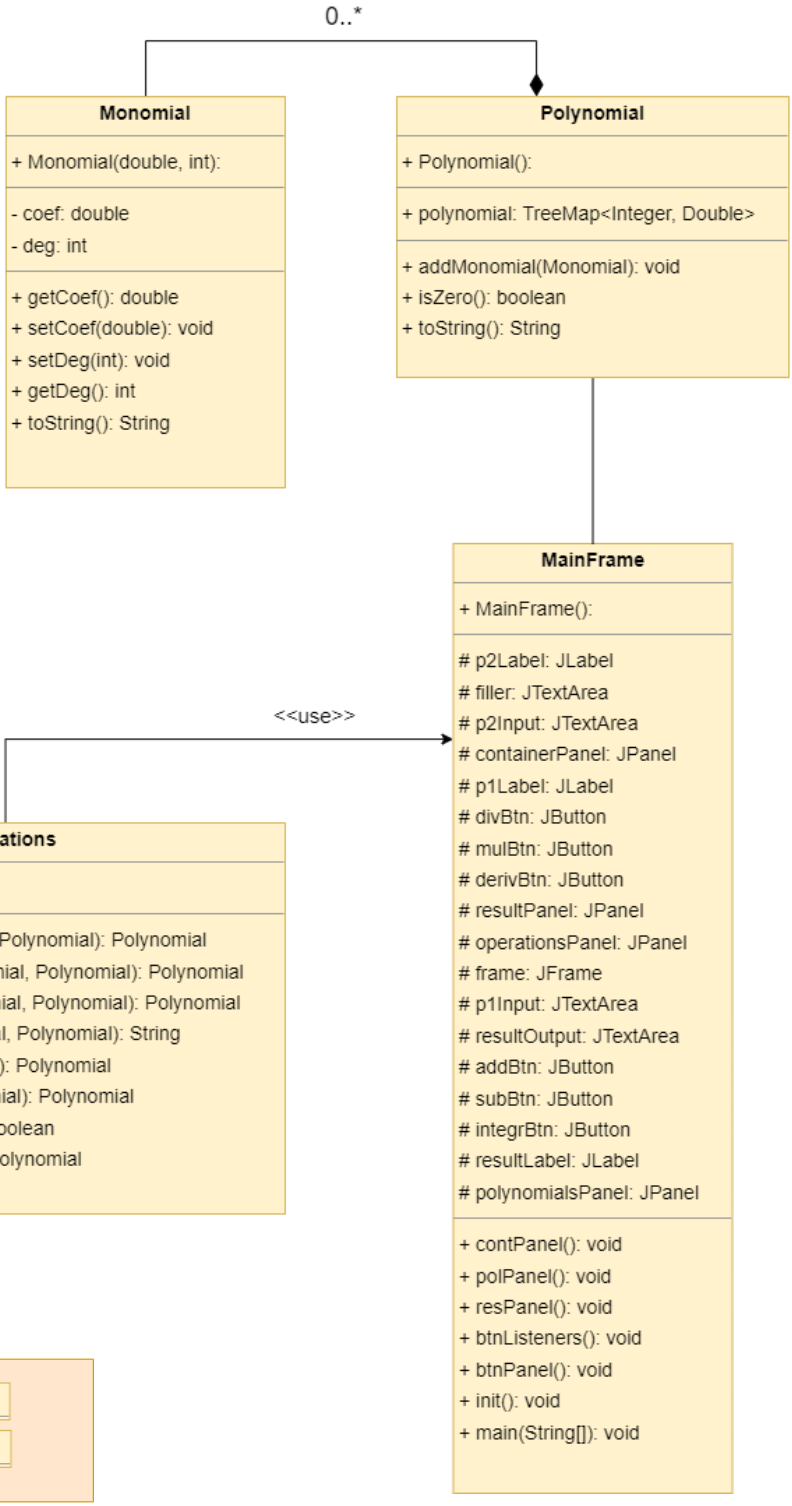
Am implementat acest sistem în Java și oferă o interfață grafică de utilizator intuitivă, care să permită utilizatorilor să introducă și să gestioneze polinoamele, precum și să efectueze operațiile dorite asupra lor. De asemenea, sistemul oferă o funcționalitate de afișare a rezultatelor într-un format ușor de înțeles și de interpretat. Beneficiile sistemului de procesare a polinoamelor sunt multiple, de la ușurarea procesului de calcul al polinoamelor și a operațiilor acestora, până la creșterea eficienței și a preciziei rezultatelor.

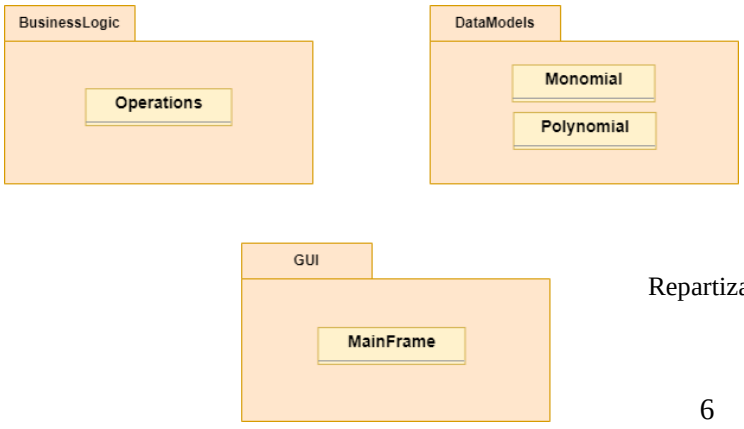
3.2 Diagrama de clase

Relația dintre clasele *Monomial* și *Polynomial* este una de compunere, unde un obiect de tip *Polynomial* conține 0 sau mai multe obiecte de tip *Monomial*.

Relația dintre clasele *Polynomial* și *MainFrame* este una de asociere, unde clasa *MainFrame* utilizează clasa *Polynomial* pentru a efectua operații pe obiectele *Polynomial* și afișarea rezultatelor.

Relația dintre clasele *Operations* și *MainFrame* este de dependență, deoarece *MainFrame* utilizează metodele (statice) din clasa *Operations*, dar *Operations* nu depinde în mod direct de *MainFrame*. În plus, *Operations* poate fi utilizat și de alte clase care au nevoie de metodele sale statice, acestea prin definiție nefiind legate de obiect instanțiat.





Repartizarea claselor în pachete

4. Implementarea

4.1 Descrierea claselor și metodelor

Datele sunt structurate sub forma a două clase:

- *Monomial*, cu atributele `coef` (coeficientul monomului), de tip **double**, și `deg` (gradul monomului), de tip **int**
- *Polynomial*, cu singurul atribut `polynomial`, un obiect de tip **TreeMap<Integer, Double>**, o clasă care implementează interfața **Map**. Aceasta oferă o modalitate de a stoca perechi *key* – *value* într-o ordine sortată bazată pe ordinea descrescătoare a cheilor, Comparator furnizat de către mine pentru stocarea polinomului descrecsător în funcție de gradul acestuia, după cum cere cerința și este, de asemenea, mai natural și ușor de citit.

În clasa *Polynomial* am implementat metoda isZero, pentru a verifica dacă polinomul e zero/nul, util în împărțire, și *addMonomial*, care adaugă un monom primit ca parametru la polinomul pe care este apelată metoda. Aceasta presupune căutarea în listă a unui element de grad egal cu monomul dat, iar la găsirea acestuia, se vor aduna coeficienții și se va modifica în monomul deja reținut. În cazul în care nu se găsește, se va adăuga pur și simplu monomul dat ca parametru.

Clasa *Operations* conține implementarea metodelor utilizate pentru efectuarea adunării, scăderii, înmulțirii, împărțirii, derivării și integrării, acestea fiind statice pentru a fi apelate și din alte clase, și pentru că nu necesită să fie apelate pe un obiect instanțiat. De asemenea, am implementat aici și *validatePolynomial*, care utilizează un *Regex* pentru a valida String-ul primit ca input de la utilizator, iar apoi, metoda *stringToPolynomial* convertește String-ul (valid – aceasta este verificată înainte de convertire) într-un obiect de tip *Polynomial*, util pentru realizarea operațiilor în sine.

`addPolynomials`

Această metodă primește două obiecte de tip *Polynomial*, reprezentând două polinoame, și returnează un nou obiect *Polynomial*, care reprezintă suma celor două polinoame. În implementarea metodei, se creează un nou obiect *Polynomial* gol care reprezintă rezultatul. Se parcurg coeficienții și gradele primului polinom, se adaugă fiecare monom în rezultat, apoi se parcurg coeficienții și gradele polinomului al doilea și se adaugă fiecare monom în rezultat. În cele din urmă, pentru a evita problemele de precizie cu numerele de tip double, coeficienții polinomului rezultat sunt rotunjiți la 3 zecimale, utilizând clasa BigDecimal. Acest lucru se face prin parcurgerea polinomului rezultat, rotunjirea valorii fiecărui coeficient (*value*), și înlocuirea valorii inițiale cu valoarea rotunjită, folosind metoda `put()` a obiectului Map.

`subtractPolynomials`

Analog metodei de adăugare, această metodă calculează diferența dintre două polinoame prin efectuarea operației de scădere între ele. Deosebirea dintre ele este faptul că primul polinom se introduce normal în polinomul rezultat, în timp ce al doilea polinom este introdus cu coeficientul negativ al fiecărui termen.

`multiplyPolynomials`

Această metodă calculează produsul dintre două polinoame transmite ca parametri și returnează un nou polinom rezultat. Pentru fiecare termen din primul polinom și fiecare termen din al doilea, se calculează gradul și coeficientul termenului rezultat din produsul celor două. Pentru a asigura o precizie adecvată a valorilor, se utilizează clasa BigDecimal pentru rotunjirea valorilor. Termenii rezultați sunt adăugați la polinomul rezultat, care este returnat la sfârșitul metodei.

`dividePolynomials`

Această metodă realizează împărțirea a două polinoame transmise ca parrametri, și returnează rezultatul sub formă de String, cu câtul (`quotient`) și restul (`rest`) concatenate.

Metoda începe prin a verifica dacă împărțitorul (polinomul al doilea) este zero sau dacă polinomul deîmpărțit (primul polinom) este zero. În cazul în care împărțitorul este zero, metoda returnează mesajul "Nu se poate face impartirea cu zero!". În cazul în care polinomul deîmpărțit este zero, metoda returnează mesajul "Quotient = 0\nRemainder = 0", deoarece împărțitorul va fi întotdeauna diferit de zero.

După aceea, se realizează inițializarea polinoamelor pentru câtul și restul împărțirii cu o valoare inițială de zero. Polinomul rest este inițializat cu valorile primului polinom, deoarece voi continua să realizez modificări pe structura lui.

Metoda apoi efectuează împărțirea dintre polinoamele unu și doi utilizând algoritmul de împărțire al polinoamelor: cât timp primul termen al polinomului rest este mai mare sau egal cu primul termen al

divizorului și ambii termeni sunt diferiți de zero, se extrage coeficientul pentru termenul cel mai mare din polinomul rest și coeficientul termenului cel mai mare din deîmpărțit, se împart acești coeficienți și se creează un nou termen pentru cât. Acest nou termen este adăugat la polinomul cât și este înmulțit cu deîmpărțitul pentru a genera un nou polinom produs. Acest nou produs este scăzut din restul curent pentru a genera un nou rest. Aceste etape sunt repetate până când restul nu mai poate fi împărțit cu deîmpărțitul sau restul devine zero. La sfârșit, metoda returnează rezultatul sub formă de String.

`derivePolynomial`

Această metodă primește ca parametru un obiect de tip *Polynomial* și calculează derivata polinomului, pe care o returnează tot sub formă de polinom. Pentru a realiza aceasta, se parcurge fiecare monom din polinomul inițial și se verifică dacă gradul monomului este mai mare decât zero, pentru că în caz contrar derivatele sunt zero. Pentru fiecare astfel de monom, se calculează noul coeficient al monomului în derivată prin înmulțirea vechiului coeficient cu gradul monomului și scăderea gradului cu 1. Această valoare este apoi adăugată ca un nou monom în polinomul de derivată. Pentru a evita problemele de precizie ale numerelor de tip double, noul polinom rezultat este rotunjit la 3 zecimale prin utilizarea clasei BigDecimal și a metodei setScale, după care este returnat ca rezultat al metodei.

`integratePolynomial`

Această metodă primește ca argument un obiect de tip Polynomial și returnează polinomul care reprezintă integrala polinomului dat ca argument. Mai precis, pentru fiecare termen al polinomului dat, se calculează noul grad al termenului prin adăugarea lui 1, iar noul coeficient se calculează prin împărțirea coeficientului vechi la noul grad. Astfel, fiecare termen al polinomului integral este dat de un nou termen cu gradul și coeficientul corespunzător, calculate în funcție de termenul corespunzător din polinomul inițial. De asemenea, metoda realizează rotunjirea coeficienților la trei zecimale, folosind clasa BigDecimal și setând scale-ul la 3, pentru a evita erorile de precizie care pot apărea în calculele cu numere zecimale.

`validatePolynomial`

Această metodă folosește o expresie regulată (*Regex*) pentru a valida un String, reprezentând un polinom. În cadrul *Regex*-ului, se caută o corespondență între șirul de intrare și un șablon specificat. Metoda compilează această expresie regulată și verifică dacă String-ul transmis se potrivește cu acesta. Dacă acesta este un polinom valid, atunci metoda va returna `true`, altfel va returna `false`.

Scurtă analiză a *Regex*-ului ales:

- `^` - începutul șirului de intrare
- `[+-]?` - semn opțional (poate fi + sau -)
- `(?:(?:\\d*\\.\\d+|\\d+)?x(?:\\^\\d+)?|\\d*\\.\\d+|\\d+)` - termen al polinomului

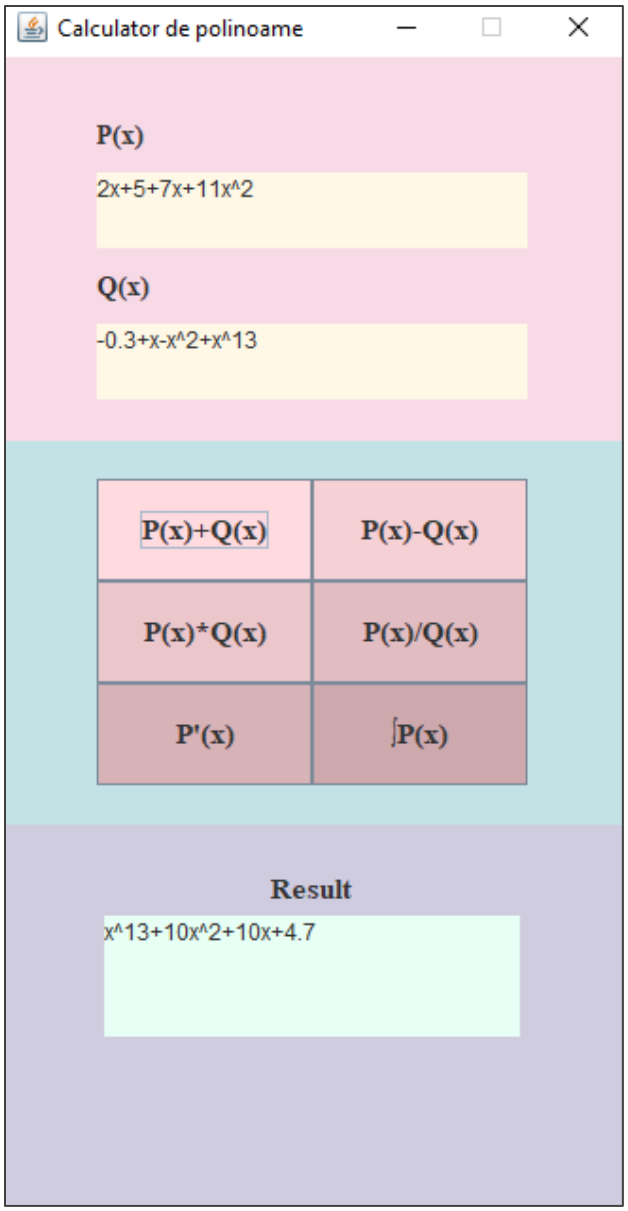
- **(?:\\d*\\.\\d+|\\d+)?** - partea coeficientului (opțională) care poate fi un număr întreg sau un număr zecimal (în care partea întreagă este opțională)
- **x** - variabila
- **(?:\\^\\d+)?** - partea exponentului (opțională) care poate fi un număr întreg pozitiv
- **|** - permite alternarea între termenii cu și fără variabilă
- **\d*\.d+** - un număr zecimal, cu partea întreagă și zecimală opționale
- **\d+** - un număr întreg

- **(?:[+-](?:(?:\\d*\\.\\d+|\\d+)?x(?:\\^\\d+)?|\\d*\\.\\d+|\\d+))*** - serie de termeni suplimentari ai polinomului, care pot fi separați de + sau -
- **$** - sfârșitul șirului de intrare

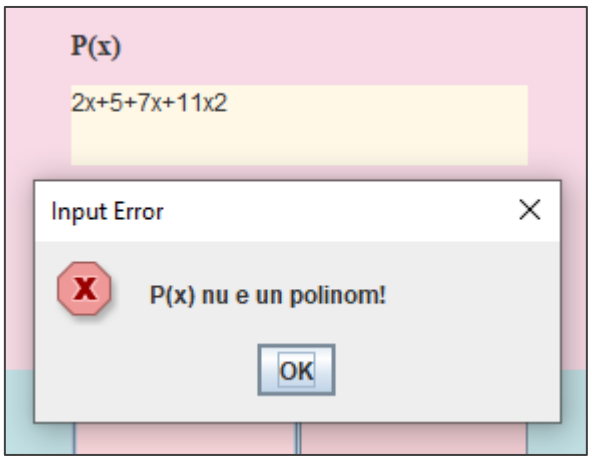
stringToPolynomial

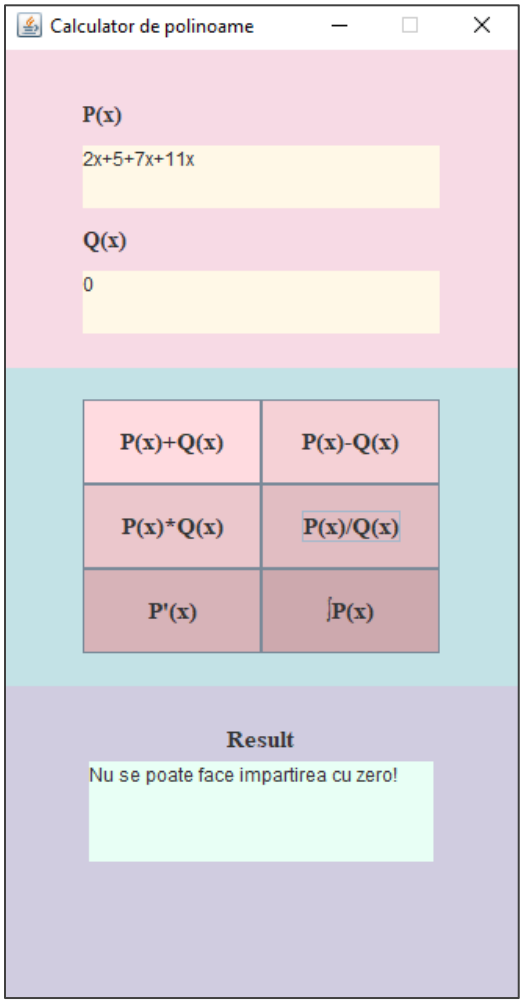
Această metodă convertește un String reprezentând un polinom (stabilit deja că e valid) într-un obiect de tip *Polynomial*. Metoda separă șirul în monoame și apoi le parsează coeficientul și gradul pentru fiecare termen, adăugându-le la polinom.

4.2 Interfața grafică



Am ales să demonstrez funcționalitatea programului meu prin aceste exemple. Se observă că putem avea coeficienți cu virgulă, monoamele nu au fost introduse ăn nicio ordine, nici a semnelor, nici a gradelor, și gradele se pot repeta. Rezultatul este formatat într-o manieră ordonată, descrescător de la gradul cel mai mare, la cel mai mic.

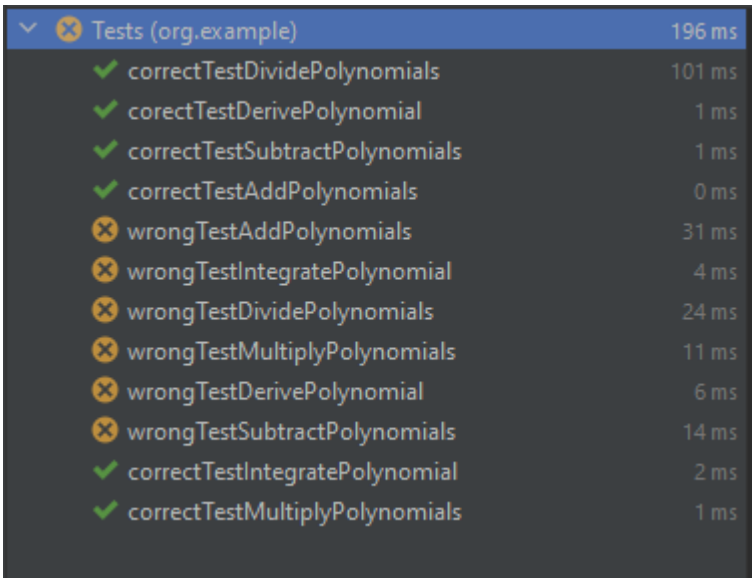




Atunci când se construiește un sistem, erorile în datele introduse pot cauza defectarea acestuia, ceea ce poate duce la oprirea sau blocarea sistemului. Pentru a evita aceste probleme, este necesară validarea datelor introduse de utilizator, deoarece nu se poate garanta că acestea vor fi introduse în mod corect.

Având în vedere importanța validării datelor și a programării defensivă, am abordat cu atenție și cazul în care utilizatorul poate introduce date eronate, atenționându-l cu un mesaj corespunzător. În prima poză, greșeala este scrierea lui 11x^2 ca 11x2, sau poate utilizatorul voia să scrie 11 + 2x. Acest mesaj de eroare atrage atenția asupra revizuirii scrierii. În a doua poză, este atenționată greșeala matematică.

5. Testare



Am realizat proiectul utilizând un archetype ***Maven*** pentru a inițializa structura proiectului, configurând dependințele și plugin-urile ***Maven*** pentru gestionarea și construcția proiectului.

Pentru scenariile de test prezentate în poză, am folosit **JUnit Test Case**, iar ca exemple am folosit polinoamele:

```
P(x) = 3x^2+5x-2
Q(x) = x+1
```

Am realizat teste și corecte, și greșite, pentru a demonstra în continuare funcționalitatea proiectului. Exemple de teste greșite sunt:

Adunarea

```
Expected :31x^2+7x
Actual   :3x^2+7x-1
```

Integrarea

```
Expected :1.9x^6+2.5x^5-2x^4-2x
Actual   :x^3+2.5x^2-2x
```

6. Concluzii și dezvoltări ulterioare

În concluzie, am luat deciziile de implementare cele mai eficiente, în limita timpului disponibil. Am acumulat multe cunoștințe noi, și aș avea câteva dezvoltări ulterioare sau îmbunătățiri.

Una din ele ar fi ideea de a putea trece de la un câmp la altul prin intermediul tastei **TAB**, de exemplu, de la polinomul 1, la 2, iar mai apoi, alegerea operației dorite, și execuția acesteia prin intermediul tastei **ENTER**. O altă dezvoltare care ar face aplicația mai generalizată și universală ar fi posibilitatea utilizatorului de a-și alege el numele variabilei. Ca ultimă idee, și una care ar putea fi cea mai utilă, este implementarea unui istoric de polinoame introduse, precum și operația realizată și rezultatul.

7. Bibliografie

- https://dsrl.eu/courses/pt/
- https://regexr.com – pentru construirea Regex-ului
- https://introcs.cs.princeton.edu/java/30oop/
- https://stackoverflow.com/questions/15625556/adding-and-subtracting-doubles-are-giving-strange-results și
- https://stackoverflow.com/questions/153724/how-to-round-a-number-to-n-decimal-places-in-java - pentru aproximarea la 3 zecimale
- https://stackoverflow.com