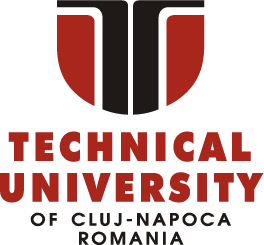
**UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA**

AUTOMATICĂ ȘI CALCULATOARE, AN II



Lucrare laborator – Primul assignment

Sistem de procesare a polinoamelor de o singură variabilă cu coeficienți întregi

Profesor curs: Prof. Dr. Ing. Cristina Pop Student: Jitaru Andrei

Profesor laborator: Teodor Petrican Grupa: 30229

**CUPRINS**

1. Introducere – Obiectivul temei
2. Analiza problemei
   1. Asumpții
   2. Modelare
   3. Scenarii
   4. Cazuri de utilizare
   5. Erori
3. Proiectare
   1. Diagrama UML
   2. Stucturi de date utilizate în implementare
   3. Proiectare clase
   4. Algoritmi
   5. Interfata Utilizator
   6. Modul de tratare al erorilor
4. Implementare
5. Testare
6. Rezultate
7. Concluzii și dezvoltări ulterioare
8. **Introducere – obiectivul proiectului**

Obiectivul acestui proiect este de a simula aplicarea diferitor operații fundamentale la nivelul polinoamelor de o singură variabilă, cu coeficienți întregi oferind posibilitatea de a ne familiariza cu diferitele caracteristici și proprietăți ale conceptelor legate de această temă. În alte cuvinte, s-a dorit implementarea în limbaj de programare Java a operațiilor de adunare, scădere, înmulțire, împărțire, derivare și integrare realizabile la nivel de polinom cu scopul de a demonstra atât înțelegerea asupra temei și cerintelor date cât și etalarea aptitudinilor de implementare a acestora în contextul limbajului de programare mai sus menționat.

Toate aspectele enumerate anterior au fost verificate prin implementarea unei aplicații simple. Aceasta este realizată sub forma unui calculator cu un aspect rudimentar (specificațiile legate de configurația grafică a proiectului vor fi prezentate în capitolul 3, sub-capitoul e) care permite introducerea de la tastatură a doua polinoame și efectuarea de operații caracteristice (adunare, scădere, înmulțire, împărțire, derivare, integrare) raportate la cele două polinoame. Conceptele legate de buna utilizare a aplicației și de posibilele cazuri de utilizare a acesteia cu scopul de a demonstra corecta înțelegere a temei și cerințelor urmează a fi prezentate în capitolul 2. A fost realizată o abordare simplă și usor de înțeles asupra implementării proiectului, abordare care va fi descrisă în cadrul capitolului 4. Corectitudinea operațiilor dar și demonstrarea rezultatelor vor fi prezentate succint în capitolele 5 și 6.

1. **Analiza problemei**

Proiectul, care în speță reprezintă un sistem de procesare a polinoamelor de o singură variabilă cu coeficienți întregi, sugerează un proces complex care necesită atenție la fiecare pas, atât în contextual realizării fiecărei operații cât și în sensul testării cât mai multor exemple diferite în scopul verificării acestora. O astfel de abordare este absolut necesară deoarece fiecare operație este unică în felul ei și prezintă diferite excepții specifice, acestea trebuind tratate în parte cu scopul obținerii unei funcționări impecabile a programului. Înainte de realizarea fiecărei operații se vor analiza anumite condiții legate de introducerea corectă a polinoamelor. Pe lângă acestea, în cazul efectuării operației de împărțire trebuiesc verificate condiții suplimentare, fapt pentru care se realizează o analiză mai detaliată a polinoamelor introduse.

1. **Asumpții**

Presupunem că în cadrul aplicației datele de intrare, și anume polinoamele,

vor fi introduse numai sub forma: anx^n + an-1x^n-1 + … + a2x^2 + a1x + a0, unde: “x” reprezintă variabila în care este reprezentat polinomul, an, an-1, … , a2, a1, a0 reprezintă coeficienții polinomului iar n, n-1, … , 2, 1, 0 reprezintă exponenții lui x. Această interpretare este considerată de către program ca fiind cea corectă. Motivele concrerte pentru care la implementare a fost aleasă respectiva reprezentăre sunt legate de faptul că la nivel general polinoamele sunt ilustrate în acest mod. Alte motive sunt legate de forma succintă și usor de înțeles a acestei reprezentări. Orice alte forme de redare a polinoamelor sunt tratate de program în momentul introducerii ca erori (vom reveni la aceste observații în subcapitolele următoare).

1. **Modelare**

Cerința temei și anume de a se realiza un sistem de procesare a polinoamelor de o singură variabilă cu coeficienți întregi poate fi implementată în diferite moduri, desigur, fiecare cu eficiența sa. Prima și cea mai banală metodă de implementare care ar putea fi abordată este tratarea polinoamelor cu ajutorul vectorilor în cadrul cărora se poate stoca atât gradul cât și exponentul acestora. Ar urma o realizare a operațiilor la nivel de vector dar și stocarea rezultatelor într-un al treilea vector rezultat din cadrul căruia se va procesa polinomul care necesită să fie convertit în String pentru a fi afișat. Desigur, rezolvarea dată mai sus nu este dintre cele mai recomandate. Această modelare a problemei prezintă diverse dezavantaje legate în special de eficiența programului, fapt pentru care s-a optat pentru o soluție mai simplă și care folosește conceptele definitorii ale polinoamelor.

Dacă privim problema din punct de vedere al construcției polinoamelor, acestea sunt constituite dintr-o înșiruire de termeni însumați între ei numiți monoame. Monoamele la rândul lor sunt alcătuite dintr-o constantă (numită coeficient), înmulțită cu o variabilă care poate avea un exponent constat întreg poztiv. Luând în considerare toate aceste aspecte s-a optat pentru crearea unui astfel de clase de obiecte care să aibă ca și atribute coeficientul și exponentul menționate mai sus. Concomitent, a fost creată clasa de obiecte “Polinom” care are ca atribut un ArrayList de monoame, în alte cuvinte: o “înșiruire” de monoame (avantajale acestei metode de reprezentare sunt diverse, începand de la proprietatea de **re-sizable array** a ArrayList-urilor față de proprietatea de **fixed-length** a vectorilor și până la faptul ca elementele dintr-un ArrayList pot fi introduse sau șterse de la o poziție particular în timp ce în cazul vectorilor ar fi necesare o serie de operații particulare).

O serie de evenimente vor fi executate de către program pentru a obține polinoamele pe care dorim să le introducem și asupra cărora vrem să efectuăm diverse calcule(pașii în detaliu vor fi prezentați în capitolele următoare). Metodele de calcul specifice operațiilor de adunare, scădere, înmulțire, împărțire, derivare si integrare au fost implementate în cadrul clasei Polinom (nu s-a optat pentru crearea unei alte clase speciale pentru operații din motive legate în mare parte de design implementării și de stilul implementatorului). Operațiile de adunare, scădere, înmulțire și împărțire presupun utilizarea a doua polinoame pentru calcule, în timp ce operațiile de derivare și integrare sunt aplicabile unui singur polinom pe rând. Toate operațiile vor returna un singur rezultat, înafară de împărțire care prezintă posibilitatea de a returna atât un quotient cât și un remainder.

1. **Scenarii**

O serie de etape trebuie urmate pentru ca aplicația să fie capabilă de a returna rezultate clare și corecte. Totalitatea acestor etape dau naștere unei serii de scenarii care definesc funcționalitatea programului. În acest sens, pentru a obține un anumit rezulat, diferite evenimente trebuie să aibă loc.

Vom presupune că datele de intrare vor fi introduse fie în JTextField-ul “Polinom1”, fie în JTextField-ul “Polinom2” - introducerea de input-uri în JTextField-ul “Rezultat” nu este o eroare însă nu este indicată deoarece nu se pot realiza operații asupra polinoamelor introduse în acesta. Alegerea operației se va realiza fie dând click pe butonul însemnat cu simbolul operației respective (în cazul adunării, scăderii, înmulțirii și împărțirii), fie prin selectarea operației din JComboBox-ul care urmează JTextField-ului corespunzator polinomului asupra căruia dorim să efectuăm operația (în cazul derivării și integrării).

Pași de utilizare ai aplicației:

* Introducerea primului polinom de la tastatură
* Dacă utilizatorul dorește aplicarea operațiilor de derivare sau integrare asupra acestui polinom:
  + Poziționarea urmată de click pe JComboBox-ul care precedează JTextField-ul în care a fost introdus polinomul
  + selectarea operației dorite din JComboBox-ul respectiv
  + afisarea rezultatului operației în JTextField-ul “Rezultat”
  + optarea pentru continuarea operațiilor sau nu
* Dacă utilizatorul dorește introducerea unui al doilea polinom pentru a realiza operații asupra ambelor date de intrare:
  + Introducerea celui de-al doilea polinom de la tastatură în JTextField-ul rămas gol (Nicidecum în JTextField-ul “Rezultat”!)
  + Alegerea operației care se dorește a fi executată prin click pe butonul dedicat operației respective
  + afișarea rezultatului operației în JTextField-ul “Rezultat”
  + optarea pentru continuarea operațiilor sau nu
* Dacă utilizatorul dorește introducerea unui al doilea polinom pentru a realiza operații asupra acestuia:
  + Poziționarea urmată de click pe JComboBox-ul care precedează JTextField-ul în care a fost introdus polinomul
  + selectarea operației dorite din JJComboBox-ul respectiv
  + afisarea rezultatului operației in JTextField-ul “Rezultat”
  + optarea pentru continuarea operațiilor sau nu

Valorile polinoamelor asupra carora dorim să efectuăm operațiile se pot modifica în orice moment.

1. **Cazuri de utilizare**

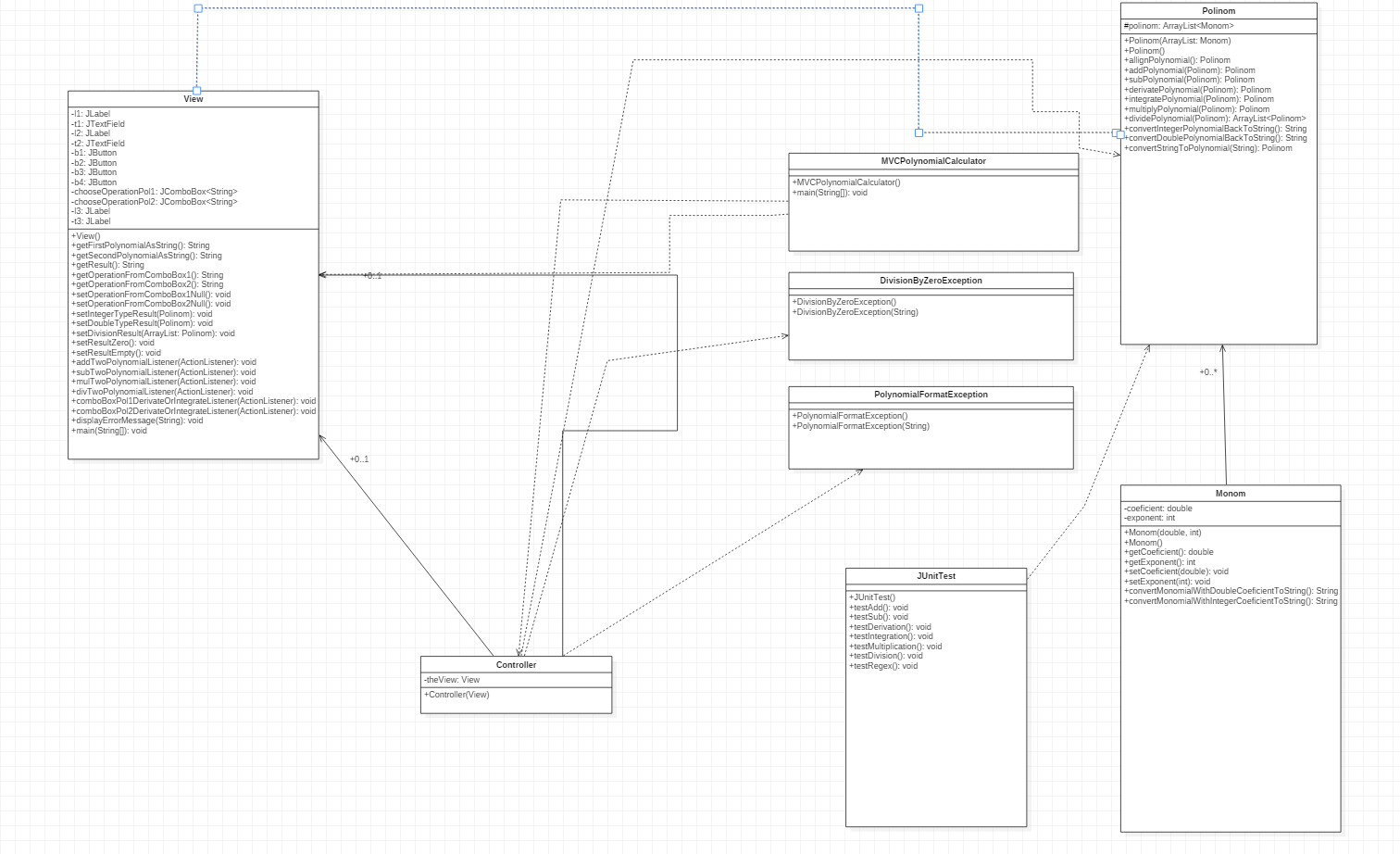
Luând în considerare faptul că acest proiect realizează rezolvarea de operații la nivel de polinoame, indiferent de polinomul introdus, și capacitatea acestuia de a returna rezultate valide, ar putea fi propus pentru a fi utilizat în domeniul învățământului, fie de către elevi, fie de către profesori. Din momemnt ce dispune de o interfă grafică simplă cu o utilizare la fel de banală, se poate considera fară lipsă de incredere ca ar putea fi folosită de orice persoană care dorește să realizeze anumite operații pe polinoame.

1. **Erori**

Anumite evenimente pot determina apariția unor excepții care au fost tratate la implementare sub forma unor erori. Apariția acestor evenimente produc afișarea unei casuțe de dialog în care este prezentată eroarea pentru a fi mai facil de identificat.

Cazuri care pot conduce la apariția erorilor:

* Dacă forma polinomului introdus nu este validă (cea prezentată în subcapitoulul “Asumpții”) atunci programul va returna o eroare de tipul “PolynomialFormatException”;
* Dacă sunt introduse caractere străine ca și input programul va returna o eroare de tipul “PolynomialFormatException;
* Dacă sunt introduse valori reale pentru coeficienți sau exponenți programul de returna o eroare de tipul “PolynomialFormatException”;
* Dacă aplicația nu primeste cel putin un set de date de intrare valid (i-a fost oferit cel putin un polinom de la tastatură) asupra caruia să execute operații specifice aplicării pe un singur polinom (derivare, integrare) atunci programul va afișa e eroare de tipul “PolynomialFormatException”;
* Dacă va fi selectată una dintre operațiile de derivare sau de integrare dintr-un JComboBox precedat de un JTextField al cărui conținut este null, atunci programul va returna e eroare de tipul “PolynomialFormatException”;
* Dacă în momentul efectuării operației de împărțire asupra a doua polinoame, împărțitorul (cel de-al doilea polinom) este polinomul null (“0”) atunci programul va returna o eroare de tipul “DivisionByZeroException”;

1. **Proiectare**
2. **Diagrama UML**

În diagrama UML de mai sus sunt prezentate toate clasele, cu atributele și relațiile dintre acestea. O analiză detaliată a acestei diagrame este prezentată în capitolul 4 al acestei documentații.

1. **Structuri de date utilizate în implementare**

Funcționarea aplicației este strâns legată de utilizarea de structuri de date în cadrul implementării acesteia. Strucuta de baza pe care iși are temeliile dezvoltarea proiectului este ArrayList-ul folosit pentru reprezentarea polinoamelor sub forma unui șir de monoame. ArrayList-ul oferă o gamă variată de posibilități de manipulare a datelor dar și a memoriei, crescând eficiența programului. Nu mai este necesară utilizarea contorizării pentru a cauta, șterge sau introduce un element din polinom, fapt pentru care vom avea acces pas cu pas la obiectele de tipul “Monom” din cadrul acestora utilizând iteratori. De asemenea alocarea dinamică a listelor reprezită un avantaj considerabil la nivel de memorie a programului.

1. **Proiectare clase**

Proiectul, ca un tot unitar, este divizat in 8 clase care permit transpunerea cerinței și temei în limbaj de programare Java. Clasele au dimensiuni, alcătuiri și roluri variate.

Întreaga aplicație a fost implementată conform modelului architectural “Model-View-Controller” utilizat în ingineria software. Alegerea acestui model se datorează izolării logicii față de considerentele interfeței cu utilizatorul, rezultând o aplicație unde aspectul vizual și nivelele inferioare sunt mai ușor de modificat, fără a afecta alte nivele. Clasa „Monom” se află la baza proiectului. Prin intermediul acesteia sunt create polinoamele care mai târziu în proces vor fi manipulate conform cerințelor. Clasa „Polinom” reprezintă „Modelul” arhitectural al proiectului, partea care manipulează operațiile logice și care se ocupă de utilizarea informației. Clasei „View” ii corespunde reprezentarea grafică a temei, în alte cuvinte exprimarea ultimei forme a datelor si anume: interfața grafică ce interacționează cu utilizatorul aplicației. Cea care realizează legatura dintre clasele „Polinom” și „View” este clasa „Controller”. „MVCPolynomialCalculator” este cea care pune cap la cap modelul MVC pentru a obține rezultatul final. Clasele „PolynomialFormatException” și „DivisionByZeroException” sunt clase care extind interfața Exception și permit personalizarea a doua excepții care să definească introducerea sub un format greșit a unui polinom și împărțirea cu polinomul nul al unui polinom. Clasa ”JunitTest” realizează teste cu scopul demonstrării corectitudinii operațiilor realizate pe polinoame.

1. **Algoritmi**

În continuare sunt prezentați principalii algoritmi care permit buna funcționare a aplicației în raport cu clasa din care fac parte:

* Clasa Monom
  + Metoda convertMonomialWithDoubleCoeficientToString() – returnează un șir de caractere care reprezintă conversia în String a unui obiect de tip Monom cu coeficienți reali;
  + Metoda convertMonomialWithIntegerCoeficientToString() – returnează un șir de caractere care reprezintă conversia în String a unui obiect de tip Monom cu coeficienți întregi;
* Clasa Polinom
  + Metoda allignPolynomial() – însumează coeficienții monoamelor cu exponenți egali și returnează un obiect de tip polinom a cărui monoame sunt ordonate descrescător în funcție de exponent. Funcționarea acestui algoritm se bazează pe cautarea continuă a monoamelor cu exponent maxim în polinomul asupra căruia se apelează metoda și crearea de noi monoame într-un nou polinom rezultat. Odată găsit un monom cu exponent maxim acesta va fi șters din polinom. Coeficientul monomului introdus în polinomul rezultat va fi egal cu suma coeficienților care au avut exponentul egal cu exponentul maxim. Exponentul maxim se va actualiza la fiecare iterație atâta vreme cât mai exista elemente în polinom.
  + Metoda addPolynomial(Polinom a) – returnează un obiect de tip polinom și primește ca argument un polinom care va fi însumat cu polinomul asupra căruia se aplică metoda. Algoritmul presupune adăugarea în lista de monoame a primului polinom a monoamelor din lista celui de-al doilea. Noua listă care conține monoamele ambelor polinoame va fi “aliniată” cu ajutorul metodei allignPolinom().
  + Metoda subPolynomial(Polinom a) – returnează un obiect de tip polinom și primește ca argument un polinom care va fi scăzut din polinomul asupra căruia se aplică metoda. Algoritmul presupune schimbarea semnelor coeficienților celui de-al doilea polinom și însumarea acestuia cu primul.
  + Metoda multiplyPolynomial(Polinom a) – returnează un obiect de tip polinom și primește ca argument un polinom care va fi înmulțit cu polinomul asupra căruia se aplică metoda. Algoritmul presupune înmulțirea fiecarui monom din primul polinom cu fiecare monom din cel de-al doilea, urmată de o “aliniere” a polinomului rezultat.
  + Metoda dividePolynomial() – returnează un obiect de tip polinom și primește ca argument un polinom care va împărți polinomul asupra căruia se aplică metoda. Algoritmul se bazează pe împărțirea Euclidiană a polinoamelor și este explicat succinct în pseudocodul următor:
* function n / d:
* require d ≠ 0
* q ← 0
* r ← n # At each step n = d × q + r
* while r ≠ 0 AND degree(r) ≥ degree(d):
* t ← lead(r)/lead(d) # Divide the leading terms
* q ← q + t
* r ← r − t \* d
* return (q, r)
* Metoda derivatePolynomial() – returnează un obiect de tip Polinom care va reprezenta rezultatul derivării polinomului asupra căruia s-a apelat metoda. Algoritmul presupune parcurgerea polinomului monom cu monom și modificarea coefiecienților și exponenților acestora conform formulelor: newCoef = oldCoef\*OldPow și newPow = OldPow-1.
* Metoda integratePolynomial() - returnează un obiect de tip Polinom care va reprezenta rezultatul derivării polinomului asupra căruia s-a apelat metoda. Algoritmul presupune parcurgerea polinomului monom cu monom urmată de modificarea coefiecienților și exponenților acestora conform formulelor: newCoef = oldCoef/(OldPow + 1) si newPow = OldPow+1.
* Metoda convertIntegerPolynomialBackToString() – returnează un obiect de tip String care va reprezenta conversia în șir de caractere a unui polinom cu coeficienți întregi. Algorimul presupune parcurgerea polinomului monom cu monom și transformarea acestora în String-uri prin apeluri ale funcției convertMonomialWithIntegerCoeficientToString() din clasa Monom.
* Metoda convertDoublePolynomialBackToString() - returnează un obiect de tip String care va reprezenta conversia în șir de caractere a unui polinom cu coeficienți reali. Algorimul presupune parcurgerea polinomului monom cu monom și transformarea acestora în String-uri prin apeluri ale metodei convertMonomialWithDoubleCoeficientToString() din clasa Monom.
* Metoda convertStringToPolynomial(String polynomialAsString) – returnează un obiect de tip Polinom și primește ca argument polinomul sub formă de String. Algoritmul presupune parsarea String-ului în Monoame aparente (care sunt de fapt niște grupuri în care se stochează coeficientul și exponentul monoamelor sub formă de String-uri) prin utilizarea unui expresii Regex. Coeficienții si exponenții vor fi convertiți din String în double respectiv int și introduși sub formă de monoame în polinomul rezultat.

1. **Interfata utilizator**

Interfața cu utilizatorul are un aspect minimalist pentru a putea fi utilizată cât mai ușor de orice persoană în parte. S-a optat doar pentru introducerea strictului necesar. Astfel, interfața conține 3 JTextField-uri, 4 butoane și 2 JComboBox-uri. JComboBox-urile au fost poziționate în dreptul JTextField-urilor în care introducem polinoamele pe care sunt oferite ca input pentru a putea înțelege asupra cărui polinom efectuăm operațiile din JComboBox-ul respectiv(derivare și integrare). Amplasarea celor 2 JTextField-uri utilizate ca input, situate simetric față de JTextField-ul corespunzător rezultatului și a celor 4 butoane cu imagini specifice operațiilor, dar și JComboBox-urilor simple și a Background-ul albastru oferă un aspect plăcut aplicației.

1. **Modul de tratare al erorilor**

După cum am menționat în capitolele anterioare există diferite evenimente care pot conduce la declanșarea de excepții pe care programul le afișeaza utilizatorului sub formă de ferestre cu erori. Acestea sunt redate în clasa Controller și sunt tratate pentru fiecare ActionListener în parte în blocuri de tipul Try-Catch. Aplicația poate să arunce doua tipuri de erori: DivisionByZeroException și PolynomialFormatException în funcție de operația efectuată și de input-ul introdus de la tastatură.

1. **Implementare**

Clasa pivot a proiectului este fără de îndoială clasa Monom. Aceasta are ca atribute coeficientul și exponentul care determină un monom (în cod sunt reprezentate cu ajutorul atributelor “coeficient” și “exponent”). Exponentul poate lua doar valori întregi fapt pentru care a fost declarat ca int, în timp ce coeficientul polinomului a fost declarat ca double deoarece, chiar dacă la intrare se acceptă doar coeficienți de tip întreg, în urma operațiilor de derivare sau împărțire se pot obține coeficienți reali. Chiar dacă cerința specifică faptul că coeficienții sunt de tip întreg programul lucrează cu coefienți reali pentru care realizează o manipulare a acestor valori pentru a putea fi afisate corespunzator pe ecran (detalii în rândurile următoare). Se dorește lucrul cu aceste atribute fapt pentru care au fost create metodele “accesoare” și “mutatoare”: getCoeficient(), setCoeficient(), getExponent (), setExponent(). Pentru a permite ca un monom să fie afișat pe ecran sub forma “anx^n” au fost create metodele convertMonomialWithDoubleCoeficientToString și convertMonomialWithIntegerCoeficientToString care returnează un obiect de tip String, prima reprezentând coeficienții sub formă reală iar cea de-a doua sub formă întreagă.

Clasa Polinom implementează ideea centrală pe care se bazează dezvoltarea temei și anume că un polinom poate fi privit ca o sumă de monoame. In acest sens clasa Polinom are ca atribut un ArrayList de monoame care materializează ideea de mai sus. Au fost implementate metodele addPolynomial, subPolynomial, multiplyPolynomial, dividePolynomial, derivatePolynomial, integratePolynomial, convertIntegerPolynomialBackToString ,convertDoublePolynomialBackToString, convertStringToPolynomial a căror funcționare și implementare a fost descrisă mai pe larg in capitolul 3, subcapitolul f.

Clasa View realizează interfața grafică a aplicației. Aceasta primește ca atribute cele 12 elemente vizuale care alcatuiesc interfața grafică: 3 JLabel-uri: l1, l2 si l3 care sunt situate în fața JTextField-urilor și au nume sugestive(“Polinom1” , “Polinom2”, “Rezultat”) pentru ca utilizatorul sa fie conștient de ceea ce introduce sau ce returnează acestea, 3 JTextFiel-uri: t1, t2, t3, două pentru introducerea polinoamelor asupra carora vrem sa efectuăm calculele (input-uri) și unul pentru afisarea rezultatului (output), 4 butoane: b1, b2, b3, b4 corespunzătoare operațiilor de adunare, scădere, înmulțire și împărțire și 2 JComboBox-uri: chooseOperațion1 și chooseOperațion2 utilizate pentru a alege dintre operațiile de derivare și integrare. În constructorul clasei a fost creat aspectul aplicației: au fost realizate 5 panel-uri: unul auxiliar pentru paginare (“aux”), unul pe care au fost înșiruite l1, t1 și chooseOperațion1, unul pe care au fost înșiruite l2, t2 și chooseOperațion2, unul pe care care au fost înșiruite cele 4 butoane b1, b2, b3, b4 spațiate unele de celelate prin suprafețe rigide și un ultim panou pe care au fost înșiruite l3 și t3. Cele 5 panel-uri au fost adaugate unui panel mai mare care la rândul său a fost poziționat pe un frame. Elementele vizuale au fost dimensionate astfel încât să obținem rezultatul dorit. Clasa View mai conține de asemenea și metode de adăugare de actionListeneri pentru JButton-urile si JComboBox-urile interfeței. Au fost create metode de getter și setter pentru a seta, respectiv pentru a extrage date din JTextField-uri și JComboBox-uri. De asemenea, a fost creată o metodă numită displayErrorMessage care creează o casuță de eroare.

Clasa Controller, cu rolul de a face legătura dintre clasele Polinom și View conține ca atribut un obiect de tip View prin intermediul căruia se realizează transpunerea grafică a datelor. Controller contine la randul ei o serie de alte 6 clase care reprezintă ActionLinster-ii pentru cele 4 JButton-uri și 2 JComboBox-uri care permit efectuarea operațiilor. În cadrul acestora sunt tratate diferite excepții (enumerate în capitolul 2, subcapitolul f) și sunt realizate interacțiunile cu interfața.

Constructorul clasei adaugă interfeței ActionListenerii menționați mai sus.

Clasa MVCPolynomialCalculator care primește ca atribut un obiect de tip View realizează prin metoda main(String[] args) punerea cap la cap a întregii aplicații.

Clasele PolynomialFormatException și DivisionByZeroException, fiecare cu 2 constructori, definesc excepțiile tratate în clasa Controller, prin transmiterea în constructori a unui mesaj specific excepției respective.

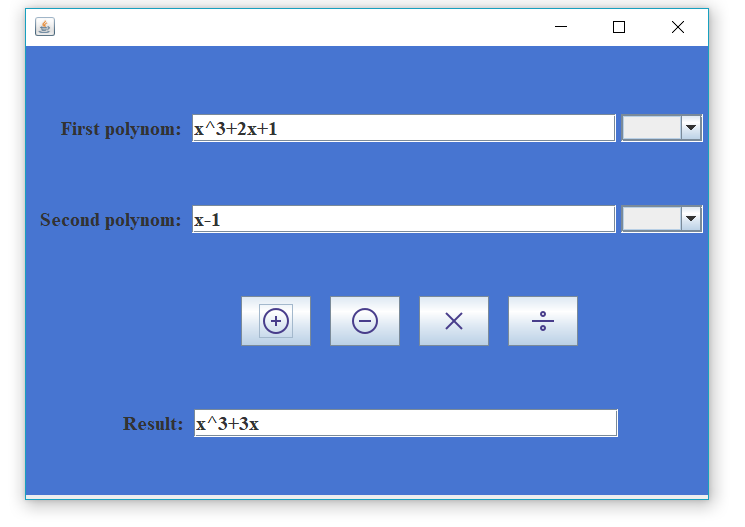
**5. Testare**

Testarea folosește Junit astfel încât fiecare metoda care definește o operație fundamentală este verificată din punct de vedere al funcționării sale corecte în clasa “JUnitTest” prin intermediu câte unui exemplu diferit. Rularea și execuția acestei clase va demonstra cu ușurința funcționarea corectă a operațiilor.

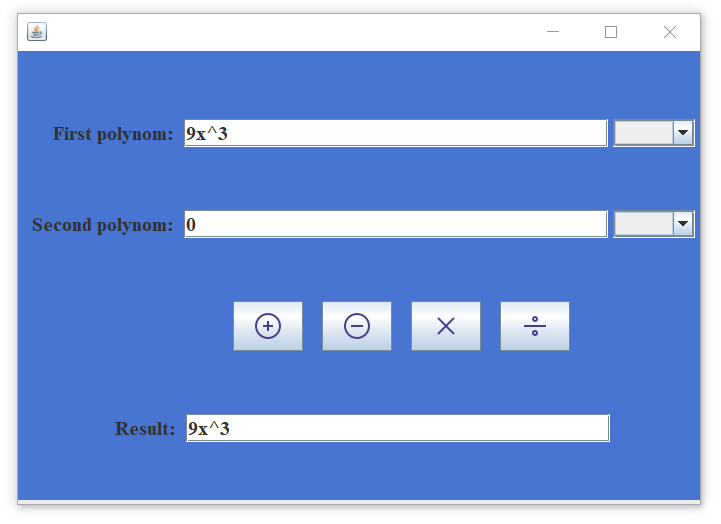
**6. Rezultate**

Au fost executate o serie de teste pentru a exemplifica diferitele rezultate care se pot obține în urma interacțiuniii utilizatorui cu interfața dar și pentru a scoate în evidență eventualele erori care pot fi afișate.

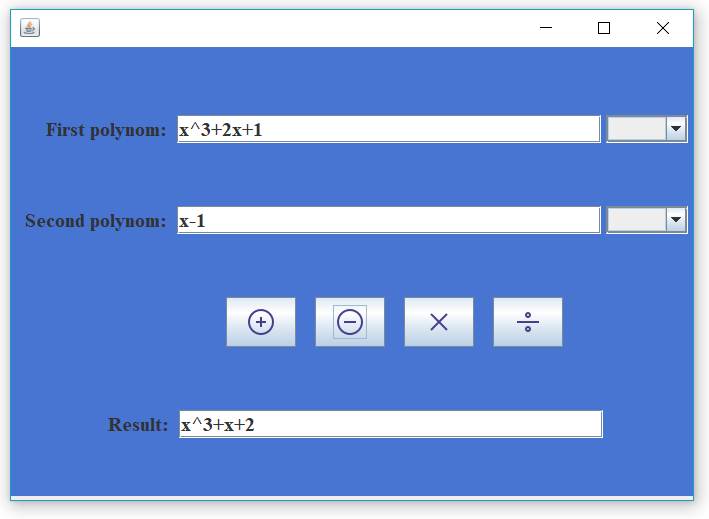
* Adunare a 2 polinoame diferite de polinomul null



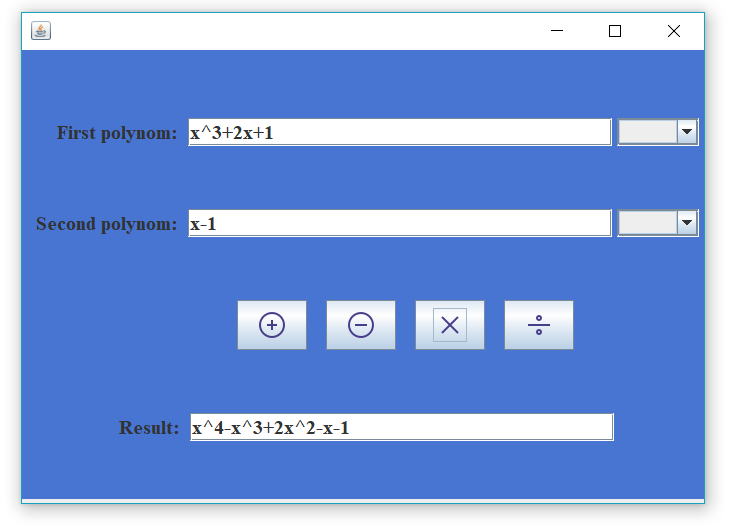
* Adunare a unui polinom cu polinomul null



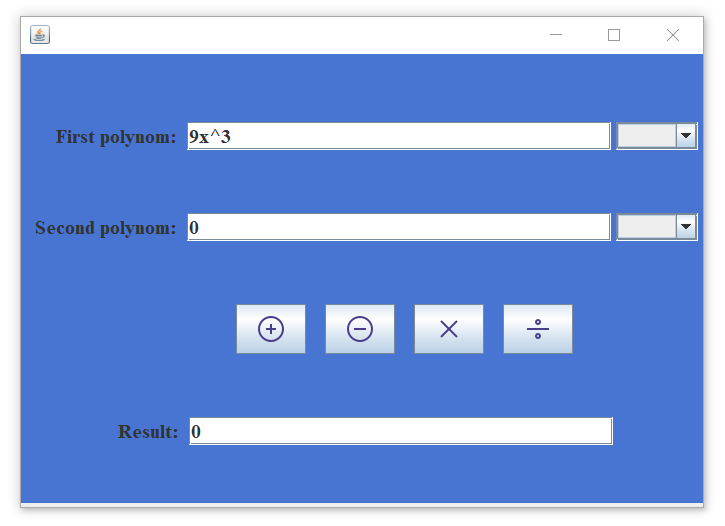
* Scădere a 2 polinoame diferite de polinom null

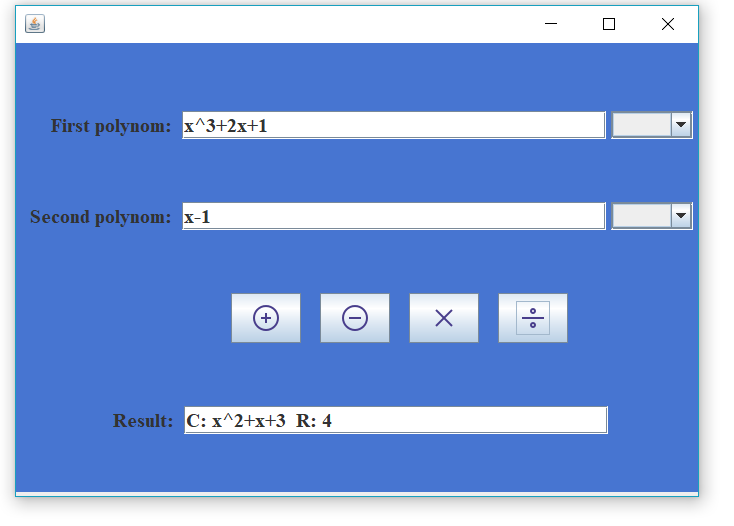
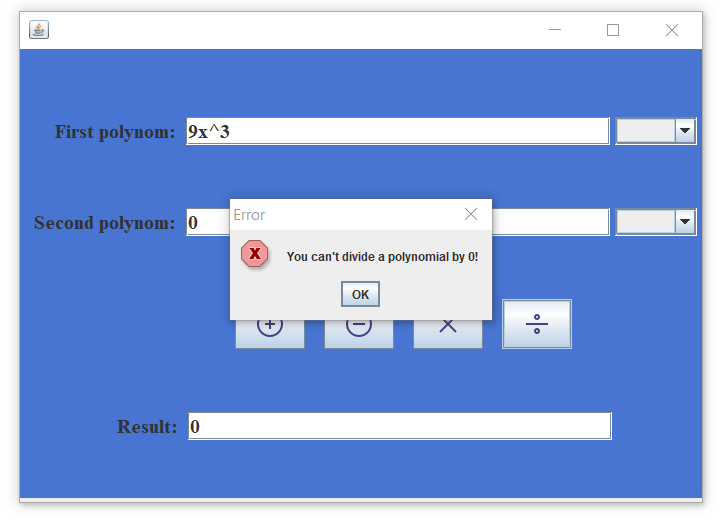


* Înmulțire a 2 polinoame diferite de polinomul null



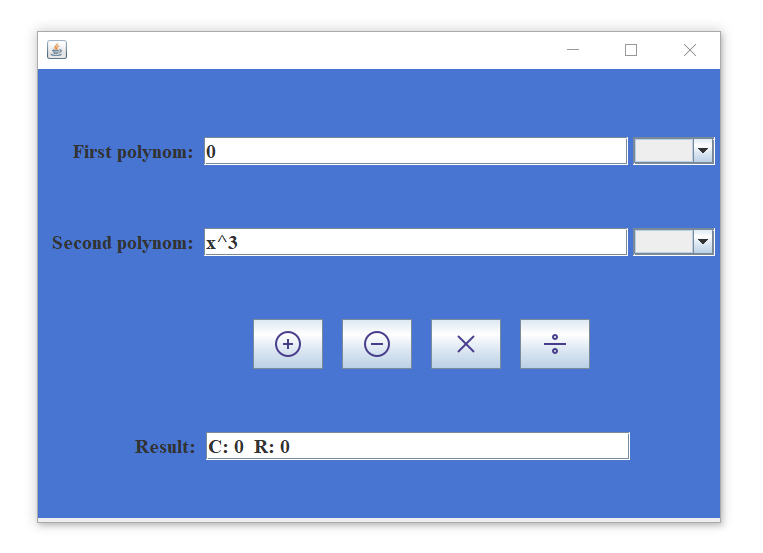
* Înmulțire a unui polinom cu polinomul null



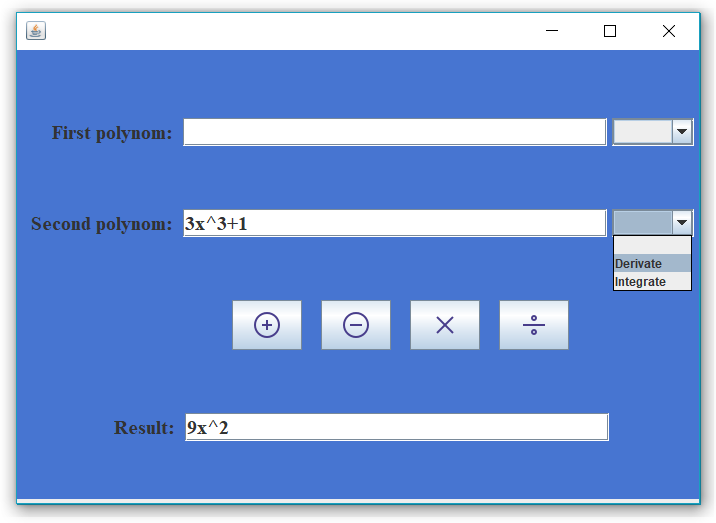
* Împărțire a doua polinoame diferite de polinomul null
* Împărțire a unui polinom la polinul null

Cazul de împărțire al unui polinom diferit de polinomul null la polinomul null genearează o eroare deorarece o astfel de operație nu este permisă.

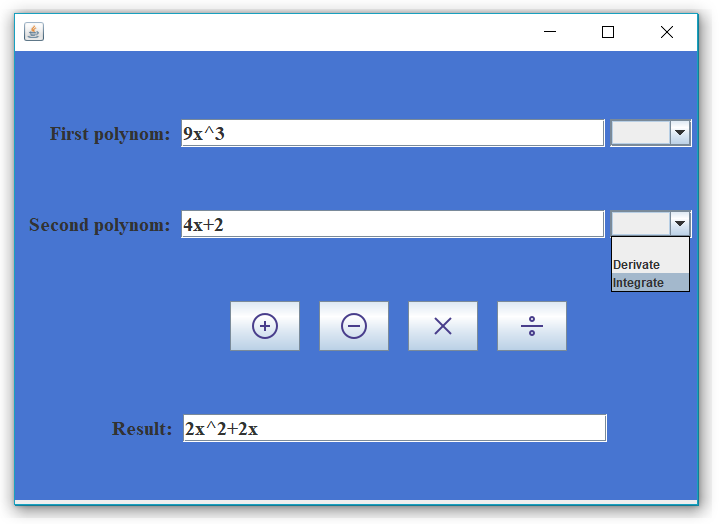
* Împărțirea polinomului null la un polinom



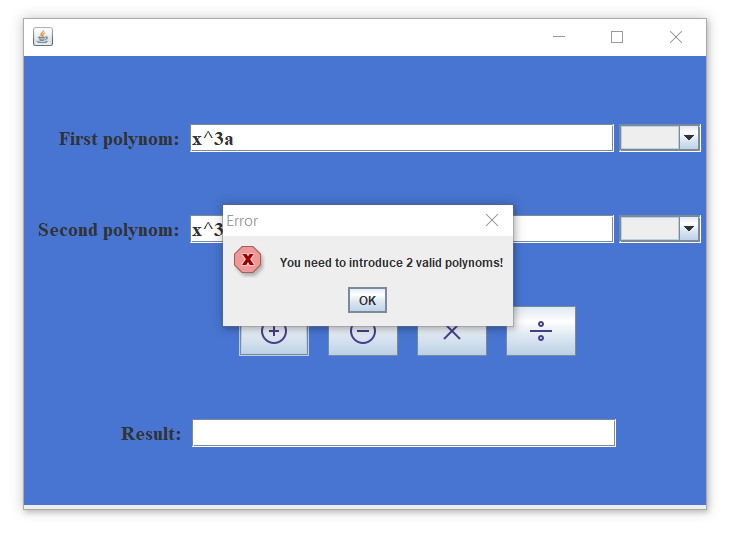
* Derivarea



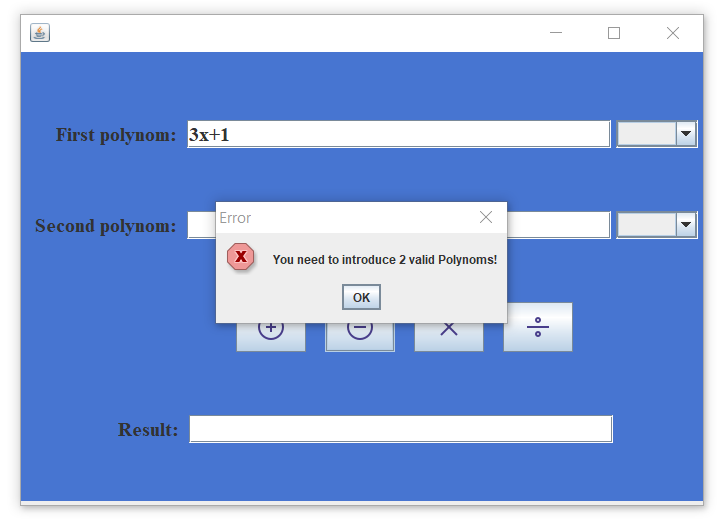
* Integrarea



* Introducerea de caractere străine



* Operații realizate cu input-uri nule



Cele 2 cazuri de mai sus, dacă se introduc caractere străine, respectiv dacă se efectuează operații cu input-uri nule, vor fi tratate similar deoarece programul va considera în ambele cazuri că input-urile introduse nu au structura unui polinom.

**7. Concluzii si dezvoltari ulterioare**

Sunt de parere ca aplicația de față prezintă posibilitatea de a fi utilizată cu usurință de absolut orice persoană interesată de realizarea de operații asupra polinoamelor de o variabilă cu coeficienți întregi. La nivel de dezvoltari ulterioare sunt propuse modificări pentru îmbunătățirea sistemului de recunoaștere a polinoamelor. De exemplu, aplicația ar putea primi un upgrade care să-i permit să accseze și polinoame de formele: an\*x^n + an-1\*x^n-1 + … + a2\*x^2 + a1\*x + a0 sau anx^n + an-1x^n-1 + … + a2x^2 + a1x^1 + a0x^0. De asemenea consider că la nivel de operații și interfașa se pot realiza îmbunătățiri în sensul că ar putea fi adăugat, de exemplu, un buton care să deschida o fereastră noua în care să fie desenat graficul unui polinom dat de la tastatură.