**TEHNICI DE PROGRAMARE**

**TEMA 1**

**CALCULATOR DE POLINOAME**

**DOCUMENTATIE**

**Nume și prenume student: Marchis Raul**

**Profesor coordonator: Pop Cristina**

**Grupa: 30229**

1. **Cerinta temei………………………………………………………………………………………….…pag 3**
2. **Obiectivul temei………………………………………………………………………………….......pag 3**
   1. **Obiectivul principal…………………………………………………………………………pag 3**
   2. **Obiectivul secundar………………………………………………………………………..pag 3**
3. **Analiza problemei, modelare, scenarii, cazuri de utilizare…………………………..pag 4**
4. **Proiectare…………………………………………………………………………………………………..pag 6**
5. **Implementare…………………………………………………………………………………………….pag 10**
6. **Rezultate…………………………………………………………………………………………………….pag 12**
7. **Concluzii……………………………………………………………………………………………………..pag 13**
8. **Bibliografie………………………………………………………………………………………………….pag 13**
9. **Cerinta temei**

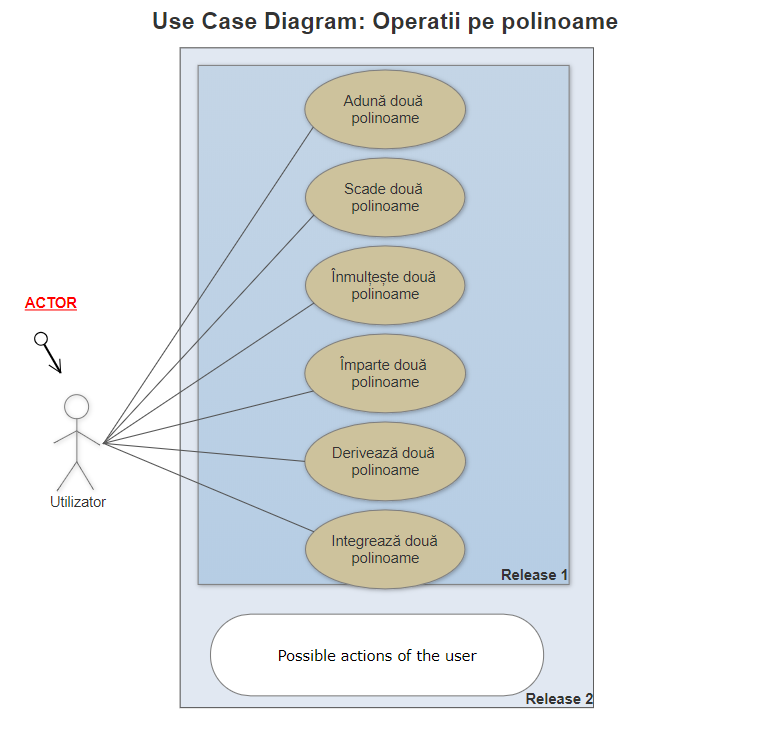
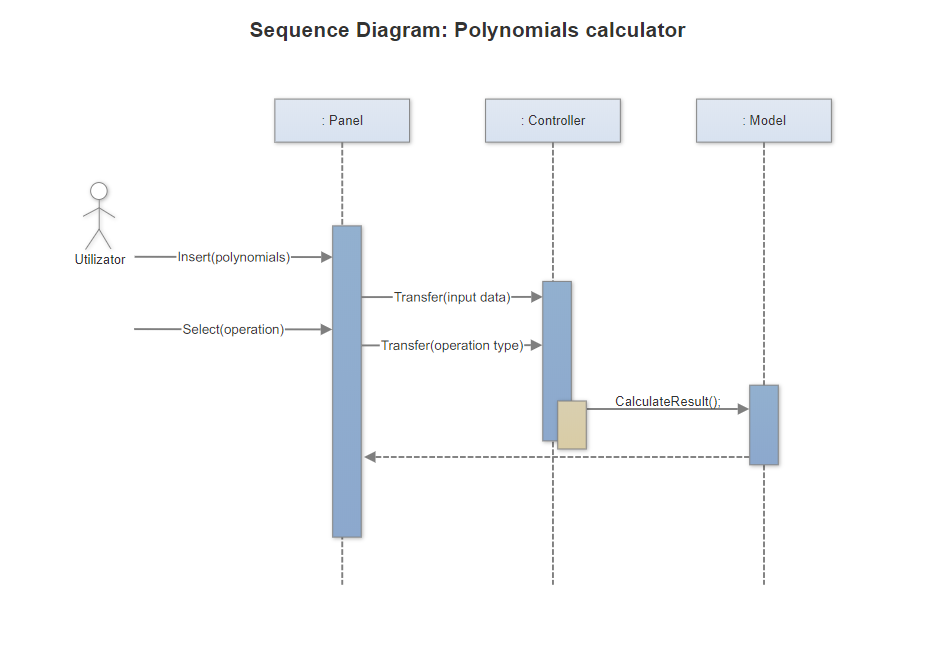
Propuneti, proiectati si implementati un sistem de procesare a polinoamelor de o singura variabila cu coeficienti intregi.

Mod de rezolvare: Se va efectua citirea polinoamelor din interfata grafica, dupa care, intrucat acestea vor fi doua string-uri, se va verifica daca introducerea lor este valida. In caz afirmativ, se vor converti in polinoame pentru a efectua operatiile posibile : adunare, scadere, inmultire, impartire, derivare, integrare. Rezultatul va fi convertit intr-un string pentru a putea fi afisat utilizatorului, in interfata grafica.

1. **Obiectivul temei**
   1. **Obiectiv principal**Obiectivul principal al temei il reprezinta implementarea unui program, cu interfata grafica, care are la baza operatii pe polinoame. Utilizatorul are libera alegere de a introduce doua polinoame si de a selecta operatia dorita din cele posibile: adunare, scadere, inmultire, impartire, derivare si integrare. Rezultatul va fi afisat dupa efectuarea calculelor.
   2. **Obiectiv secundar**

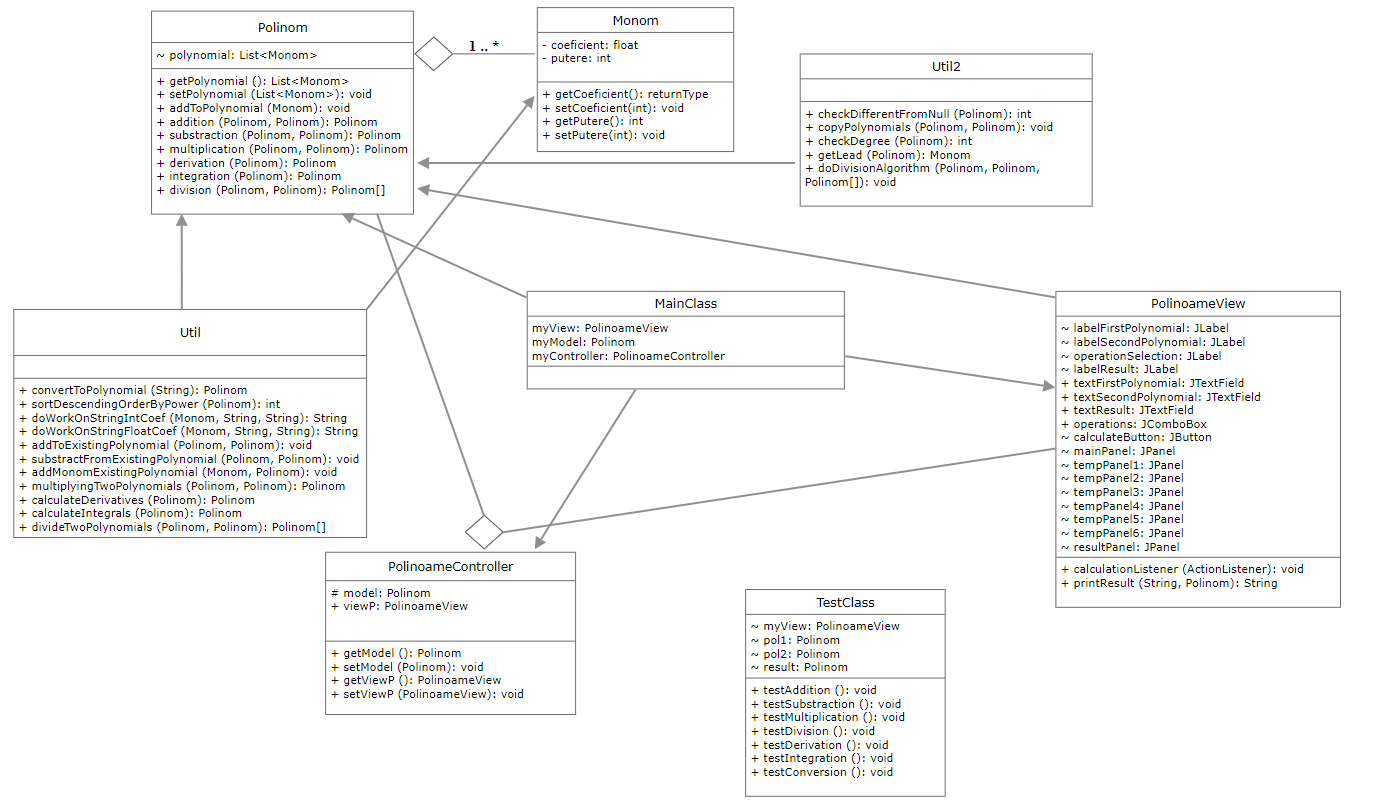
|  |  |  |
| --- | --- | --- |
| Obiectivele secundare | Descrierea obiectivelor | Capitolul |
| Diagrame si descrieri de use-case | Se vor propune diferite cazuri posibile de introducere a datelor sub forma unui flow-chart si tratarea lor. | Capitolul 2 |
| Structurile de date folosite | Se vor explica structurile de date utilizate si din ce motiv au fost alese acestea. | Capitolul 3 |
| Deciziile de proiectare / implementare | Se va explica modul de gandire ales atat pentru interfata grafica cat si pentru implementare. | Capitolul 3 |
| Algoritmi utilizati | Se vor descrie algoritmii implementati pentru preluarea datelor, parsarea string-urilor in polinoame, respectiv algoritmii folositi pentru fiecare operatie in parte. | Capitolul 3 |
| Pachete folosite | Se va descrie modul de implementare **Model-View-Controller,** respectiv organizarea aplicatiei pe pachete si explicatia aferenta. | Capitolul 3 |
| Testarea aplicatiei / rezultate | Se va explica modul de testare al aplicatiei cu ajutorul **Junit.** | Capitolul 5 |
| Concluzii | Se vor trage concluziile si vor fi subliniate lucrurile invatate in urma temei. | Capitolul 6 |

1. **Analiza problemei, modelare, scenarii, cazuri de utilizare**Use-case pentru operatiile: adunare, scadere, inmultire, impartire  
   **1. Utilizatorul(actorul) introduce cele doua polinoame**  
    Normal flow: se introduc cele doua polinoame valide, dupa structura prezentata ca exemplu in interfata grafica: exemplu “-2+3x^3-2x”.   
    Bad flow: utilizatorul introduce polinoame invalide, spre exemplu: “-2+3x^3+ab”.  
    Tratarea **bad flow-ului** / cazuri speciale: se va verifica string-ul introdus de utilizator in ambele campuri, iar in cazul in care acestea nu respecta structura prezentata, se va afisa un mesaj de eroare prin care utilizatorul va fi informat despre acest lucru.  
   **2. Utilizatorul(actorul) selecteaza operatia dorita**  
    **Normal flow**: este selectata o operatie posibila din cele sase, aflate in combo box-ul interfetei grafice.  
    **Bad flow / cazuri speciale**: avem cateva cazuri speciale, explicate mai jos **Tratarea bad flow-ului / cazuri speciale** :   
    - In cazul in care utilizatorul introduce ambele polinoame corect, insa al  
    doilea polinom este “0” si acesta selecteaza operatia de impartire, se va afisa un   
    mesaj de eroare intrucat nu este permis acest lucru.  
    - In cazul in care este selectata operatia de derivare sau integrare, ea va fi   
    efectuata doar pentru primul polinom introdus, iar utilizatorul va fi informat  
    despre acest lucru.  
    - In cazul operatiei de integrare sau impartire, rezultatul cel mai probabil  
    va contine coeficienti reali, fapt pentru acestia vor fi afisati cu doua zecimale.  
    **3. Utilizatorul apasa butonul “Calculate”, afisat in interfata grafica**  
    **Normal flow**: rezultatul este afisat in text field-ul alocat.  
    **Bad flow / cazuri speciale**: -  
    **Tratarea bad flow-ului / cazuri speciale :** cazurile speciale si tratarea lor a fost  
    atinsa la punctul 2.  
   **4. Utilizatorul poate selecta o alta operatie, pe aceleasi date de intrare(polinoame)** **Normal flow**: utilizatorul selecteaza, spre exemplu, operatia de adunare a doua  
    polinoame, dupa care doreste si scaderea lor. Rezultatul afisat se modifica.  
    **Bad flow / cazuri speciale**: utilizatorul selecteaza, spre exemplu, operatia de   
    integrare.  
    Se calculeaza rezultatul, dupa care acesta selecteaza operatia de adunare.  
    **Tratarea bad flow-ului / cazuri speciale**: nu exista un bad flow, intrucat   
    este la libera alegere a acestuia ce fel de operatii doreste sa efectueze.  
    Un caz special ar fi urmatorul: dupa ce utilizatorul selecteaza operatia de   
    integrare sau impartire, se va calcula rezultatul si se va afisa in interfata.  
    Al doilea camp pentru introducerea polinomului, va fi suprascris cu un   
    mesaj prin care acesta va fi informat despre faptul ca, in cazul derivarii   
    si a integrarii, rezultatul se va calcula bazat pe primul polinom. O a doua  
    operatie selectata(adunare, scadere, inmultire, impartire) va genera o eroare  
    intrucat se va incerca transformarea string-ului din al doilea field, intr-un  
    polinom. Acesta este un caz special la nivel de implementare, pe care personal  
    l-am tratat prin inlocuirea text-ului din al doilea field cu “0”, in cazul selectiei  
    altei operatii.

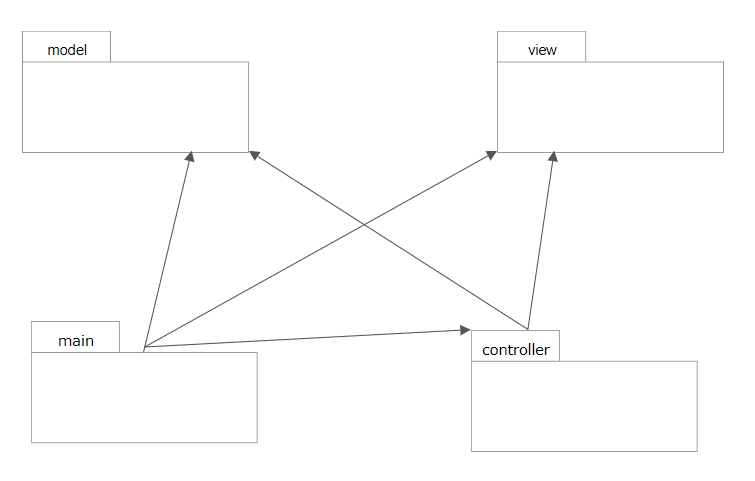
  
  
  


1. **Proiectare**

**Decizii de proiectare** Pentru proiectarea aplicatiei, am decis sa lucrez dupa modelul **Model – View – Controller**. Astfel, intreaga aplicatie este “impartita” in trei mari pachete, despre care se va discuta intr-o subsectiune urmatoare.   
  
**Proiectare clase** Personal, m-am gandit sa incep intai cu “cea mai interesanta” parte a proiectului pentru un utilizator, adica interfata grafica. Astfel, am creat o clasa **PolinoameView** ce contine implementarea interfetei cu utilizatorul. GUI-ul este simplu implementat, dar ofera detaliile necesare astfel incat user-ul sa respecte introducerea corecta a datelor si sa lucreze cu ele.  
Clasa contine o singura metoda, pentru a adauga ascultator pe buton-ul “Calculate”.  
 Clasele **Monom** si **Polinom** sunt necesare pentru a converti cele doua polinoame introduse de utilizator, care sunt extrase sub forma de String-uri (un Polinom este alcatuit din mai multe Monoame). Tot in clasa Polinom, se vor implementa metodele aferente fiecarei operatii in parte, dar doar o parte din ele, intrucat algoritmii fiecarei operatii au fost scrisi separat in cele doua clase de ajutor : **Util** si **Util2.** M-am gandit din perspectiva unui proiect mare : daca cineva ar trebui sa lucreze vreodata mai departe pe acest proiect si sa-l imbunatateasca, ultimul lucru de care ar fi interesat ar fi algoritmii pentru adunare si scadere si ce se intampla in spatele lor. Astfel, metodele aferente operatiilor din clasa Polinom sunt foarte scurte(maxim 5-6 linii de cod), insa intreaga implementare(algoritmii) a lor se afla in cele doua clase (totodata, in acest mod fiecare metoda va avea maxim 30 linii de cod scrise, rezultand astfel un cod usor de inteles si concis).   
 Deoarece am explicat sensul claselor **Util** si **Util2**, as vrea sa vorbesc putin despre metodele implementate in interiorul lor si anume : convertirea string-urilor in polinoame, sortarea unui polinom in ordine descrescatoare a puterilor monoamelor, prelucrarea afisarii polinomului rezultat in urma unei operatii precum si “tot ceea ce se intampla in spatele unei adunari, scaderi, inmultiri, etc” a doua polinoame. Modul de gandire al acestor operatii vor fi explicate si ele.  
 Intrucat clasele **PolinoameView** si **Polinom** comunica intre ele si au nevoie de schimb de informatii, am implementat clasa **PolinoameController** pentru a ajuta in aceasta privinta. Controller-ul extrage informatiile din interfata grafica, le trimite model-ului pentru a le prelucra, iar rezultatul returnat de model este mai apoi trimis din nou interfetei grafice pentru a-l afisa.  
 Clasa **TestClass** este o clasa pentru a testa aplicatia, insa mai multe detalii vor fi prezentate in capitolul 5.  
  
**Diagrama UML de clase**



**Diagrama UML de pachete**

****

**Structuri de date** In ceea ce priveste structurile de date folosite, am decis sa folosesc LinkedList pentru a reprezenta un polinom (o lista de monoame). Stiind ca are deja implementate toate metodele necesare, am considerat va fi mult mai practic si mai usor de lucrat cu acestea: iterator(), hasNext(), next(), set(), add(), etc.

**Pachete folosite** Dupa cum spuneam, am hotarat sa lucrez dupa structura **“Model—View—Controller”.** Aplicatia este astfel alcatuita din trei pachete, pentru a distinge responsabilitatile fiecarei clase si legatura lor cu clasele din acelasi pachet.   
 **1. View** – pachetul contine o singura clasa, si anume clasa **PolinoameView,** intrucat este deajuns pentru a implementa intreaga interfata grafica. Clasa respectiva mosteneste clasa JFrame, astfel ea insasi reprezinta un frame pe care asezam containere (panel-uri).   
 **2. Model** – este format din patru clase : **Monom**, **Polinom**, **Util** si **Util2**. Acestea se afla in stransa legatura una cu cealalta deoarece sunt concepute in acelasi scop: transformarea String-urilor in polinoame si capacitatea de a efectua operatii. Dupa cum spuneam si in subcapitolul “**Proiectare** **clase**”, pentru a mentine lungimea metodelor sub 30 de linii de cod, cat si pentru a mari lizibilitatea codului, am hotarat ca implementarea efectiva si “tot ce se intampla in spatele unei operatii” sa se desfasoare in acestea. Metodele ajutatoare din aceste doua clase, **Util** si **Util2** pot fi accesate de oriunde intrucat au fost facute statice, tocmai in acest scop.  
 **3. Controller –** pachetul este alcatuit dintr-o singura clasa, ce reprezinta principala legatura intre model si view. Controller-ul extrage informatiile introduse de utilizator in interfata grafica, le trimite model-ului pentru a fi prelucrate, dupa care se efectueaza operatia selectata. Rezultatul este din nou prelucrat pentru a putea fi afisat sub forma de String, dupa care este trimis view-ului.   
 Clasele **MainClass** si **TestClass,** deoarece sunt clase separate si nu se inrudesc cu niciuna de mai sus, vor fi puse intr-un pachet separat.  
 Astfel, lucrand dupa structura “**MVC**”, chiar daca in viitor aplicatia va fi prelucrata si imbunatatita, la adaugarea unor lucruri noi nu se va pune niciodata problema “unde ar trebui sa implementam metoda respectiva?”, deoarece toate clasele inrudite fac parte din acelasi pachet, deci au responsabilitati asemanatoare.

**Algoritmi** **Adunarea –** prin intermediul metodei **addToExistingPolynomial** din clasa **Util**, parcurg intai primul polinom, in paralel cu un polinom rezultat (la inceput 0). Pentru fiecare monom din primul polinom, parcurg intreg polinomul rezultat pentru a vedea daca exista un anumit monom cu aceeasi putere, iar in caz afirmativ, le adun. In cazul in care nu a fost gasit un monom cu aceeasi putere in polinomul rezultat, adaug monomul pur si simplu. Acelasi lucru are loc si pentru al doilea polinom, in paralel cu rezultatul (care deja contine primul polinom), astfel ca rezultatul final nu va contine doua monoame cu aceeasi putere. Rezultatul este sortat in ordinea descrescatoare a puterilor.

**Scaderea –** prin intermediul metodei **addToExistingPolynomial** din clasa **Util**, se parcurge primul polinom pentru a fi adaugat la rezultat. Pe urma, folosind metoda **substractFromExistingPolynomial** din aceeasi clasa **Util,** parcurgem cu primul iterator polinomul in care se va afla rezultatul final (in care momentan, se afla doar primul polinom) in paralel cu al doilea polinom. Cand se gasesc doua monoame cu aceeasi putere, din monomul din rezultat se scade monomul din al doilea polinom, iar in cazul in care puterea e unica, se va adauga la rezultat cu semn schimbat (inmultire cu (-1)).

**Inmultirea –** folosind metoda **multiplyingTwoPolynomials** din clasa **Util**, se vor parcurge cele doua polinoame cu doi iteratori, in paralel. Fiecare monom din primul polinom va fi inmultit cu fiecare monom din al doilea polinom, iar rezultatul va fi adaugat la rezultat prin intermediul metodei **addMonomToExistingPolynomial**, din aceeasi clasa. Aceasta metoda primeste doi parametrii, un Monom si un Polinom (rezultatul) si verifica daca exista deja puterea respectiva. In caz afirmativ, se aduna cele doua monoame sau se adauga monom-ul cu putere unica la rezultat.

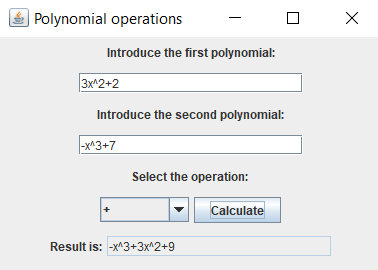
**Impartirea** – algoritmul pentru impartire este cel clasic: atata timp cat rang-ul deimpartitului este mai mare sau egal cu rang-ul impartitorului, continuam sa efectuam operatia de impartire. Se imparte monomul cu cel mai mare rang din deimpartit la monomul cu cel mai mare rang din impartitor. Rezultatul impartitii se adauga la cat. Pentru a afla restul impartirii facem in urmatorul mod : la inceput de algoritm, copiem deimpartitul intr-un polinom numit rest. Dupa fiecare impartire, din rest se scade rezultatul impartirii inmultit cu impartitorul. Asadar, acestea constituie o prima iteratie in algoritm. In cazul in care restul continua sa aiba rang-ul mai mare sau egal cu impartitorul va avea loc o a doua iteratie, o a treia iteratie si asa mai departe pana cand rang-ul rest-ului este mai mic decat rang-ul impartitorului. La finalul algoritmului, vor rezulta doua polinoame: unul ce contine rezultatul impartirii (catul) si altul ce contine restul impartirii. Metoda pentru impartire se numeste **divideTwoPolynomials**, aflata in clasa Polinom, iar metodele aditionale pentru cautarea rangului spre exemplu, se afla in clasa **Util**.

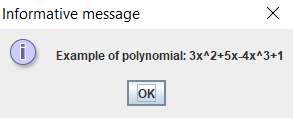
**Derivarea** – derivarea unui polinom presupune prelucrarea fiecarui monom in parte: se inmulteste puterea curenta a monomului cu coeficient-ul si se decrementeaza puterea cu 1. Cazul special este derivarea unui monom cu rangul 1: acesta devine o constanta, reprezentata de coeficientul acestuia. Metoda care efectueaza derivarea se numeste **division** in clasa **Util**, iar metoda ajutatoare se numeste **calculateDerivatives** din clasa **Util**.

**Integrarea** – integrarea unui polinom presupune acelasi lucru ca derivarea: prelucrarea fiecarui monom in parte. Noul coeficient al termenului integrat va fi dat de coeficientul monom-ului impartit la puterea incrementata cu 1 (newCoeficient = oldCoeficient \* (power+1)). Puterea noului monom rezultat in urma integrarii o reprezinta vechea putere incrementata cu 1 (newPower = oldPower + 1). Metoda care efectueaza integrarea se numeste **integration** in clasa **Polinom**, iar metoda ajutatoare se numeste **calculateIntegrals** in clasa **Util**.

**Parsarea string-urilor si construirea polinomului** – algoritmul presupune utilizarea metodei implicite Split din clasa String. Mai intai, se inlocuiesc toate spatiile libere din polinomul introdus de utilizator, dupa care se imparte string-ul dupa caracterele “+” si “-“. Pe urma, fiecare string rezultat in urma acestui lucru va fi din nou separat dupa caracterele “x” si “^” pentru a afla coeficientul si puterea monomului respectiv. Se construieste un monom cu puterea si coeficientul gasit dupa care se adauga in lista de monoame a polinomului respectiv.

1. **Implementare** In principiu, metodele importante au fost discutate in subsectiunea **algoritmi**, asadar voi descrie **implementarea interfetei utilizator.**



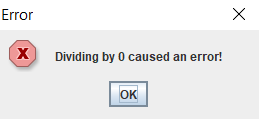
****

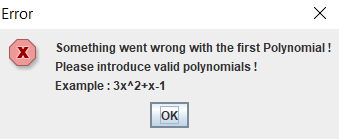
M-am gandit sa fac o interfata grafica simpla, dar din care utilizatorul sa inteleaga ce are de facut.

Odata cu aparitia interfetei grafice, am afisat si un mesaj informativ prin care utilizatorul poate vedea un exemplu de introducere a polinoamelor.

Asadar, legat de implementarea GUI-ului, am instantiat trei label-uri pentru a oferi mai multe detalii in ceea ce priveste actiunile pe care user-ul le are de facut pentru ca aplicatia sa functioneze: el trebuie sa introduca doua polinoame, unul in primul text field si celalalt in al doilea text field. Dupa selectarea operatiei dorite, odata cu apasarea butonului **Calculate**, in dreapta label-ului “Result is” se va afisa rezultatul operatiei respective, intre acele doua polinoame. Exceptiile si tratarea cazurilor nefavorabile vor fi prezentate in cele ce urmeaza:

**Impartire cu 0**

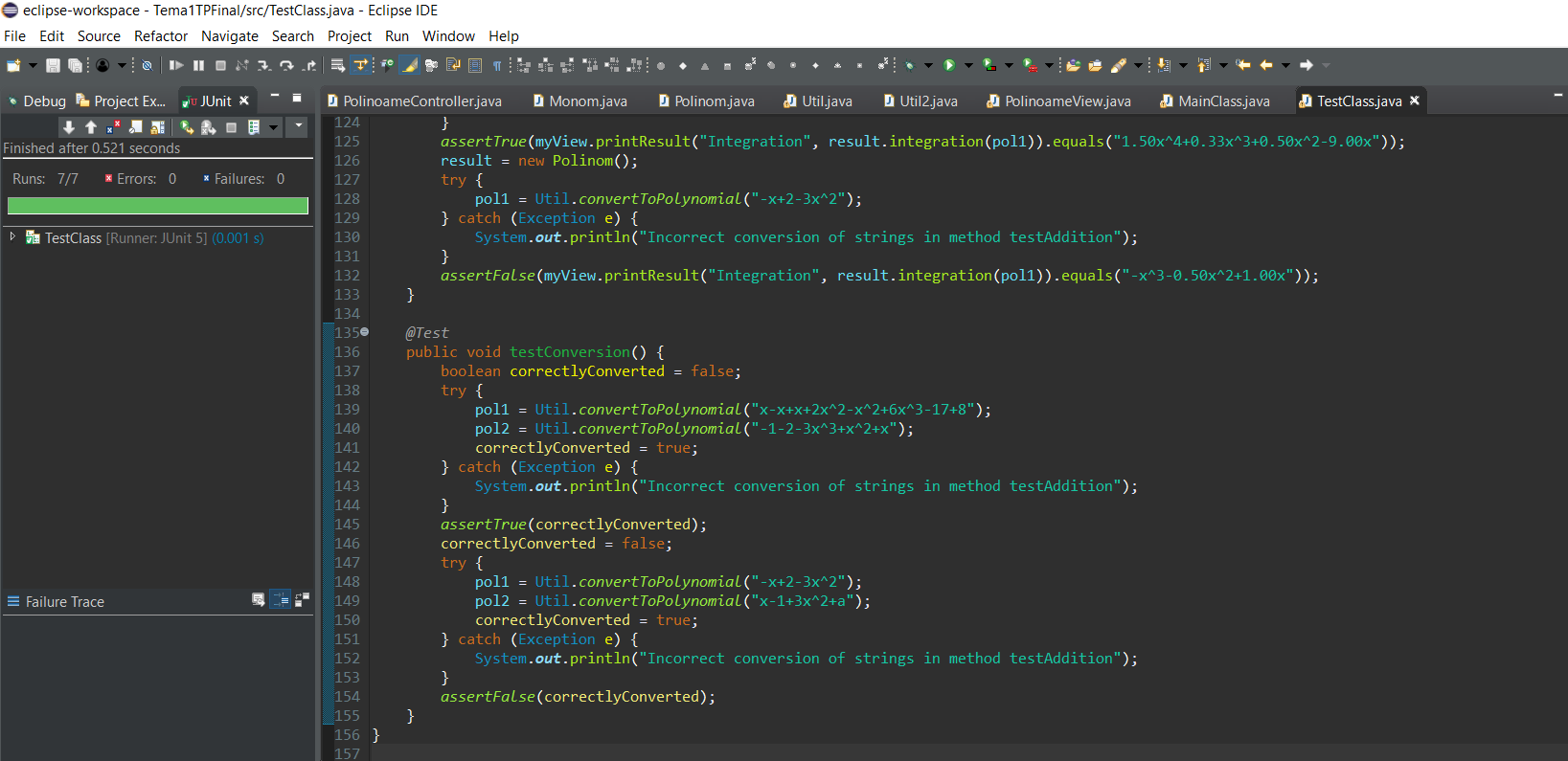


**Introducerea gresita a datelor (polinoamele)**  
  


Interfata grafica contine un singur ascultator, care a fost pus pe butonul **Calculate**. Acesta fiind apasat, se incepe executia aplicatiei: sunt extrase datele introduse in cele doua field-uri, se incearca transformarea lor in polinoame, dupa care are loc calcului si afisarea acestuia. Erorile intampinate pe parcurs vor fi semnalate printr-un mesaj de eroare.

1. **Rezultate**

Pentru a ma asigura ca aplicatia functioneaza cum trebuie si ca rezultatele sunt calculate corect, am implementat cu ajutorul **JUnit** cazuri de testare pentru diferitele functionalitati ale acesteia.  
 Pentru fiecare operatie in parte, am propus doua cazuri de testare: un assertTrue() si un assertFalse(), astfel se verifica din ambele puncte de vedere daca rezultatul operatiei este calculat cum trebuie (la assertFalse, rezultatul prezis fiind foarte aproape de rezultatul corect, eventual mai mare cu 1).



1. **Concluzii**

In urma acestui proiect, lucrul cel mai important pe care l-am invatat il reprezinta tratarea tuturor cazurilor speciale, intrucat in interactiunea cu utilizatorul, nu putem lasa bug-uri in aplicatie sau cazuri care sa pluteasca in aer.

1. **Bibliografie**

Generarea diagramelor : [www.smartdraw.com](http://www.smartdraw.com)  
Pseudocod pentru impartirea polinoamelor : <https://en.wikipedia.org/wiki/Polynomial_long_division>