**Universitatea Tehnica din Cluj-Napoca**

Calculatoare, anul 2

**LUCRAREA 1**

**CALCULATOR DE POLINOAME**

**Bogdan Stupariu, grupa 30425**

**1. Introducere**

**1.1. Obiectivul lucrării**

Lucrarea urmarește implementarea unei aplicații plecând de la urmatoarea cerință : „Propuneți, proiectați și implementați un sistem de procesare a polinoamelor de o singură variabilă cu coeficienți intregi.”. Astfel, cerința presupune construirea unui sistem care să producă operații pe unul sau două polinoame. Polinoamele pe care se produc operații sunt polinoame cu coeficienți întregi.

**1.2. Abordarea lucrării**

Această lucrare prezintă o metodă de rezolvare a problemei enunțate, incluzând o serie de operații ce pot fi aplicate pe polinoame: Adunare, scădere, înmulțire, împărțire, integrare si derivare. Au fost alese aceste operații deoarece sunt cele mai uzuale si întâlnite operații între polinoame

**2. Descrierea problemei**

**2.1. Analiza problemei**

Analiza problemei constă în parcurgerea unei serii de obiective cu scopul final de a întelege modul de implementare și de funcționare al programului. Din start este clar că proiectul trebuie să conțină două parți.

Prima parte este reprezentată de interfată prin care user-ul interacționează cu programul. În acest caz, user-ul trebuie sa aiba posibilitatea de a introduce polinoame și să aleagă operația pe care vrea să o aplice. Deoarece orice operație de n polinoame poate fi redusă la multiple operații de două polinoame, am ales ca user-ul sa poate introduce doar două polinoame. Dimensiunea polinoamelor este limitată de memorie, însă în lucrarea de față nu vom considera un obiectiv operarea pe polinoame cu puteri mai mari de 109. După introducerea polinoamelor, user-ul trebuie să aibă posibilitatea de a alege operația pe care vrea să o efectueze. După efectuarea operației, utilizatorul trebuie să poată citi rezultatul ușor.

Din matematică știm că structura unui polinom este

P(x) = CnXp + Cn-1Xp-1 + … … … + C1X + C0

Una dintre problemele cele mai mari care au apărut pentru implementarea proiectului a fost stabilirea unui mod de introducere a datelor de către utilizator, astfel încat în cea de-a doua parte a proiectului, datele introduse de către user să fie întelese de către compilator.

A doua parte este reprezentată de către funcționalitate, partea ce nu este vizibilă de către utilizator, ci doar de către programator. Această parte trebuie să citească informațiile transmise de către user și să opereze astfel încat rezultatul transmis către utilizator să fie corect. Prima etapă va fi legată de citirea datelor introduse și convertirea lor într-un format care să fie organizat și ușor operabil de către utilizator. Odată citită informația și convertită, programul trebuie să selecteze operația dorită de către utilizator și să o ofectueze în funcție de datele introduse. Operațiile prezentate mai sus sunt operații clasice.

Cele mai dificile probleme apărute s-au aflat la nivelul modului de conversie a datelor astfel încât orice ar introduce un utilizator de la tastatură, programul să returneze un rezultat corespunzător. Daca user-ul introduce date ce nu sunt polinoame, cum ar fi două cuvinte, programul trebuie să redea către user un mesaj corespunzător. O alta problemă intâmpinată a fost în cadrul implementării operațiilor, respectiv la nivelul operației de împărțire. Algoritmul în sine, dar și posibilitatea de a avea rezultate reale, nu întregi, sunt două probleme ce au necesitat o analiză profundă.

**2.2. Modelare**

Odata analizată problema, următorul pas din proiect este modelarea, conform principiilor de OOP. Definirea claselor nu a reprezentat o problema, structura fiind una clara. În primul rând, era necesară existența unei clase care să funcționeze ca interfață între user și program. În al doilea rând, știm că proiectul conține polinoame, iar fiecare polinom conține unul sau mai mulți termeni. Am ales implementarea fiecăruia în câte o clasă. De asemenea, avem o serie de operații, fiecare implementată în câte o clasă. Necesitatea respectării principiilor OOP a făcut mult mai ușoară alegerea structurilor de date și a metodelor de lucru.

**2.3. Scenarii**

Scenariile de funcționare ale proiectului sunt bazate pe inputul user-ului. În cazul în care inputul este cel cerut de indicații, proiectul va urma calea aleasă de user și va returna rezultatul dorit de către acesta. În cazul în care inputul nu corespunde cerințelor, programul nu va performa operații și va returna un mesaj corespunzător astfel încât user-ul să poată cunoaște eroarea.

Scenariile de proiectare multiple pe care le-am luat în calcul au fost, în special, legate de interfața grafcă, de modul de prelucrare a datelor și de operațiile folosite. Daca la nivel de operații folosite cerința îmi oferea posibilitatea de a alege între integrare/derivare, grafic și aflare de rădăcini și astfel a fost mai mult o alegere personală, la nivel de prelucrare a datelor a trebuit să construiesc mai multe scenarii pentru a vedea care este cea mai buna metodă de implementare. În acest caz am analizat 2 metode principale: prelucrarea String-ului folosind RegEx (Regular Expressions) sau prelucrarea folosind algoritmi de parsare. Implementand în paralel ambele scenarii, am ales folosirea RegEx deoarece, chiar daca este o metodă mult mai greu de înteles, mi-a dat impresia unei maleabilități și toleranțe la cazuri aparte mult mai bune.

**2.4. Cazuri de utilizare**

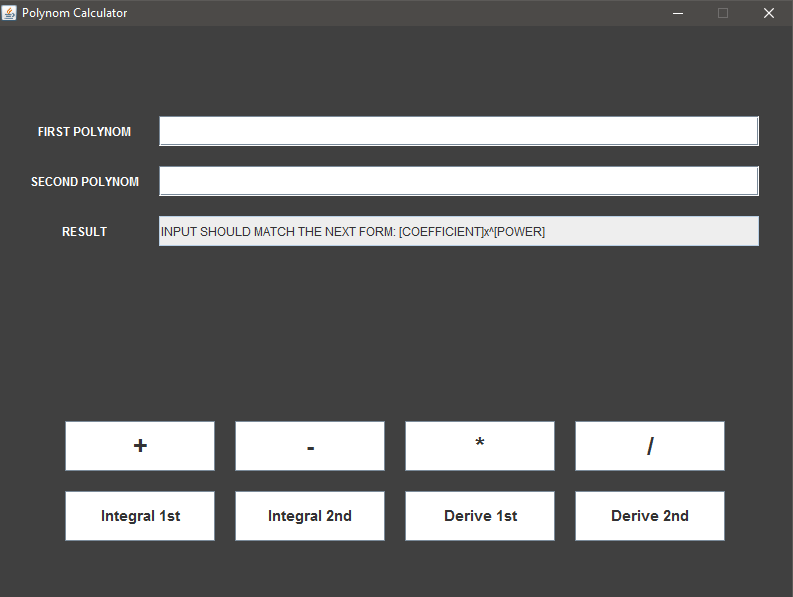
La acest capitol ne referim la modul în care relaționează user-ul cu programul. Pentru a îmi da seama cum să implementez această relație, am stabilit următoarele obiective:

a) Interfața trebuie să fie operațională

b) interfața trebuie să fie contruită pe principiul „user-friendly”

c) interfața trebuie să conțină informații referitoare la modul în care user-ul trebuie să interacționeze

d) interfața trebuie să prezinte un aspect plăcut.



Imagine 1. Interfața grafica - GUI

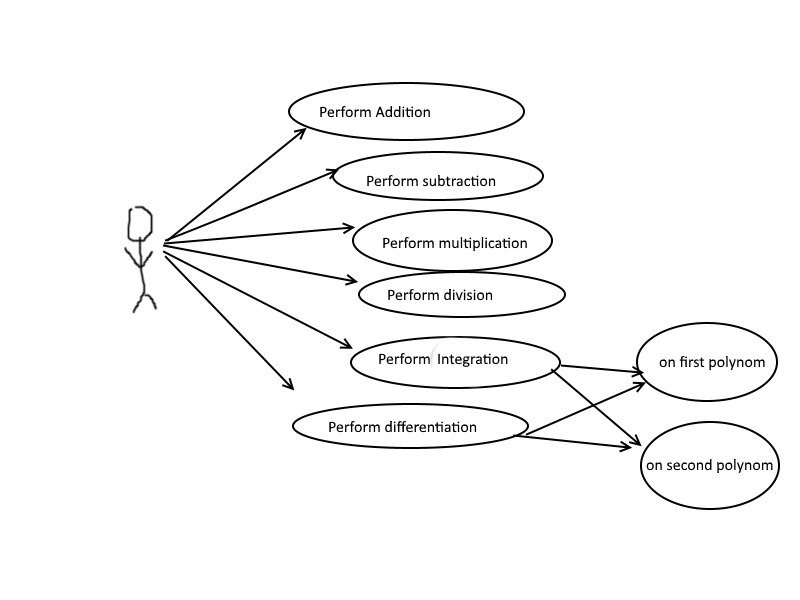
User-ul va introduce polinoamele dorite în spațiile destinate. Aceste spații, în interiorul programului sunt definite ca JTextField. Alegerea unei operații (definite ca JButton) va declanșa o acțiune și textul va fi citit. Textul String va fi preluat din JTextField, prelucrat printr-un algoritm RegEx și vor fi construite polinoamele, definite ca ArrayLists de termeni. În funție de acțiunea declanșată, una dintre clasele de operații va fi apelată. În cadrul acestui apel, polinomul polyResult va fi construit. Odată finalizată operația, polinomul va fi transformat înapoi în format String și va fi transmis către cel de-al treilea JTextField, cu denumirea de RESULT.

Este important de precizat că, în funție de operația aleasă, user-ul este nevoit sa introducă ambele sau doar unul dintre polinoame. În cazul operațiilor aflate pe primul rând, user-ul este nevoit să introducă ambele polinoame, fiind operații care se aplică pe minim două polinoame. În caz contrar, programul va genera mesajul „INCORRECT INPUT”. În cazul operațiilor aflate pe cel de-al doilea rând, user-ul poate introduce un singur polinom, operația fiind aplicată pe acel polinom. Insă, user-ul trebuie sa selecteze operația ce este aplicată pe polinomul introdus. Spre exemplu, dacă user-ul introduce un polinom în primul spațiu, acesta va putea să aplice operațiile „Integral 1st” și „Derive 1st”, în caz contrar va primi mesaul de „INCORRECT INPUT”.

**3. Proiectare**

**3.1. Diagrama UML**

**3.1. Use Case Diagram**



Use Case Diagram relevă relația dintre user și program. User-ul poate utiliza programul accesând unul dintre următoarele cazuri, ilustrate în *Figure 2:* Performarea adunării, scăderii, multiplicării și împărțirii. De asemenea, acesta poate performa integrarea și derivarea pe primul sau pe al doilea polinom. Astfel, sunt create 8 cazuri de funționare ale programului.

**3.1.2. Class Diagram**

**3.1.3. Sequence Diagram**

**3.2. Structuri de date**

Structurile de date cele mai importante din proiect sunt cele de tip ArrayList. Practic, prin acestea sunt reținute în cadrul proiectului polinoamele și sunt efectuate operațiile. ArrayList<Term> este o lista formată din multiple clase de tip Term, ce conțin un coeficient Coef și o putere Power. Înlănțuirea mai multor instanțieri de tip Term formează un polinom.

O altă alegere de implementare ce merită menționată, chiar daca nu este o structură de date în sine, este enumerația IDOp, ce conține toate tipurile de operații ce sunt folosite. Prin acestă structură, este implementată selectarea operației ce trebuie să fie efectuată. Fiecărei operații îi este atribuită în momentul declarării ActionListener-ului corespunzător un IDOp definit în enumerație. În cadrul clasei Polynom, apelarea clasei de operație corespunzătoare se face în funție de IDOp-ul transmis.

**3.3. Proiectarea Claselor**

Acest punct se referă la modul în care a fost modelat proiectul din punct de vedere al claselor, bazat pe principiile OOP. Se iau în vedere împărțirea pe pachete, clasele conținute de fiecare pachet sau interfețele utilizate. Inițial, am pornit cu ideea de a avea 2 pachete, unul care să conțină interfața cu utilizatorul, GUI, iar celălalt funționalitatea programului. Odată începută analiza problemei și am elaborat diagramele, am coștientizat că partea de operații face parte dintr-o categorie distinctă și că astfel, necesită gruparea într-un pachet separat, implementat sub numele de „mathOperations”. Apariția celui de-al 4-lea pachet se datorează separării claselor Polynom și Term de clasa main ce asigură funționalitatea programului.

**3.3.1. Pachetul run**

Pachetul run conține două clase, Main și Const.

* Clasa Main este clasa principală a programului și apelează clasa GUI pentru a lansa programul.
* Clasa Const este o clasă de constante ce conține toate constantele programului. Chiar daca în cazul acestui program nu era necesară construcția acestei clase, am decis introducerea ei pentru a aplica un principiu de bună practică. Consider că, în orice proiect OOP, este foarte utilă construirea unei clase ce conține constantele din proiect, pentru a facilita accesul la modificarea lor. De asemenea, consider și un avantaj din punct de vedere al securității datelor.

Acesta conține două constante declarate, ambele definind dimensiunile Frame-ului

*public final static int HEIGHT = 600;*

*public final static int WIDTH = 800;*

**3.3.2. Pachetul gui**

Pachetul gui conține o singură clasă, GUI, ce implementează interfața programului. Voi explica funționalitatea metodelor folosite în această clasa. Pentru o întelegere mai bună a designe-ului, am ales adăugarea unei diagrame:

* public GUI() este motodă constructor a clasei, folosită pentru instanțierea clasei GUI din clasa Polynom, fără a creea un nou frame (ca în cazul instanțierii prin folosirea constructorului apelat din clasa Main.
* *public GUI(int HEIGHT, int WIDTH)* este metodă constructor a clasei, apelată în clasa Main.

În această clasă sunt apelate metodele buildFrame(), buildUpPanel(), setHelp() și buildDownPanel() și practic creeaza întreaga interfață a programului

În afara metodelor apelate, în interiorul acestei clase sunt definite principalele JPanel ale GUI, respectiv mainPanel, ce creează cadrul interfeței și oferă fundalul dark grey, contentPanel, ce pregătește secționarea interfeței prin utilizarea GridBagLayour, respectiv upPanel și downPanel, ce împart interfața în două zone ce, practic, vor deservi la implementarea JtextField-urilor (pentru upPanel), respectiv a Jbutton (pentru downPanel).

* private void buildFrame(int HEIGHT, int WIDTH) este metoda ce setează atributele pentru frame-ul creat, JFrame myFrame.
* public void buildUpPanel(JPanel upPanel) este metoda folosită pentru crearea jumătății superioare a interfeței grafice. Layout-ul setat este GridBagLayout, iar în cadrula acestei metode este apelată metoda createInput(). Sunt dezvoltate JtextField-urile definite la începutul clasei.
* private void createInput( . . . . . ) este o metodă folosită pentru simplificarea codului și evitarea refolosirii multiple a codului. Prin fiecare metodă de acest fel apelată, sunt definite structurile de tip Jpanel ce indică utilitatea fiecărui JtextField
* private void setHelp() este apelată pentru a indica în JtextField-ul corespunzător rezultatului modul în care user-ul trebuie să introducă polinoamele pentru a fi validate.
* private void buildDownPanel(JPanel downPanel) construiește jumătatea inferioară a interfeței grafice, din care fac parte cele 8 butoane ce declanșează apelul operațiilor. Modelul de layout ales este de GridBagLayout. În cadrula cestei metode este apelată metoda createButton().
* private void createButton( . . . . . ) este o metodă folosită pentru simplificarea codului și pentru evitarea refolosirii multiple a codului. Prin fiecare metodă de acest fel apelată, sunt definite structurile de tip Jbutton.
* setSecondString( . . . . ), setFirstString( . . . . ) și setResult ( . . . . ) sunt metode de tip setter folosite pentru setarea conform ghidului de bune practici OOP ale inputului și outputului. secondString și firstString sunt structuri folosite pentru a memora informația luată din JtextField, iar setResult( ) setează conținutul resultBox, JtextField-ul ce reprezintă rezultatul operației.
* public void actionPerformed(ActionEvent e) este o metodă generată prin implementarea de către clasa GUI a ActionListener și asigură funționalitatea butoanelor. Prin această metodă, în momentul în care un buton este apăsat de către user, se apelează constructorul clasei Polynom transmițându-i-se String-ul citit (firstString, secondString) și tipul de operație corespunzătoare, prin folosirea enumerației. Apelarea metodelor setFirstString și setSecondString în debutul metodei asigură recitirea informațiilor introduse de către user de fiecare dată când o nouă operație este aleasă.

Am ales acestă implementare din două motive. În primul rând, un rezultat este așteptat doar după ce o operație este aleasă de către utilizator. În al doilea rând, este necesară recitirea intrărilor de fiecare dată când o operație este apelată deoarece user-ul poate modifica intrările între două aperări succesive.

**3.3.3. Pachetul polynom**

Pachetul polynom conține două clase, Polynom și Term și este foarte important pentru structura programului. Cele două clase conținute definesc modul în care datele sunt prelucrate și reținute în cadrul programului pentru a putea performa operațiile.

**a) Clasa Term** definiște termenul unui polinom. Dacă P(x) = CnXp + Cn-1Xp-1 + … … … + C1X + C0

este un polinom, atunci CnXp este un termen al polinomului. X fiind variabilă, în cadrul termenului trebuie reținute constantele, respectiv coeficientul și puterea termenului. Puterea este declarată ca integer, iar coeficientul este declarat double pentru a putea efectual într-un mod cât mai corect algoritmul de împărțire.

* În cadrul acestei clase, singurele metode întâlnite sunt cele de tip Set și Get pentru coeficient și putere și așa cum am prezentat și anterior, reprezintă un comportament de bune practici OOP și susțin securitatea programului

**b) Clasa Polynom** este clasa care, în opinia mea, creează cea mai importantă parte din funționalitatea programului și a necesitat cea mai multă analiză. Clasa definește trei polinoame (firstPolynom, secondPolynom și resultPolynom) implementându-le ca ArrayList de Term. Pentru aceasta, preia stringurile citite în clasa GUI și le modelează astfel încât să creeze termeni. Termenii sunt adăugati listei ce formează polinomul. Tot în cadrul clasei sunt apelate și operațiile matematice.

* public Polynom(String firstString, String secondString, IDOp opType) este metodă constructor apelată de GUI în cazul operațiilor efectuate pe două polinoame. În cadrul acestei metode se instanțează constructorul GUI() pentru a folosi setterul setResult. Metodele conținute vor fi prezentate în cele ce urmează.
* public Polynom(String string, IDOp opType) este constructor identic cu cel precedent, însă este apelat în cazul operațiilor efectuate pe un singur polinom
* private void parseString(String str, ArrayList<Term> poly) este metoda ce asigura conversia de la sir de tip String la ArrayList de termeni. Cu ajutorul Regular Expressions, sirul este impartit in siruri partiale, delimitate de semnul plus sau minus. Sirurile create sunt adăugate într-un șir și este apelată metoda regexChecker().
* private void regexChecker(String theRegex, String text, ArrayList<Term> poly) este metoda ce modifică fiecare subșir în termen și îl adaugă în ArrayList. Totul este realizat cu ajutorul RegEx, ce stabilește criterii clare pe baza cărora sunt formați termenii.

*Implementarea acestei metode a reprezentat cea mai complicată etapă a implementării proiectului. Chiar dacă existau alternative, am considerat oportun studiul asupra RegEx, aflând importanța și folosirea extinsă a acesteia în diferite domenii (între timp, cunoștiintele acumulate s-au dovedit utile până și la construirea formularelor de tip GoogleForms).*

* private void unparseString(ArrayList<Term> result) este metodă folosită pentru conversia din ArrayList în String a polinomului. În acestă metodă, cea mai importantă structură de date este StringBuilder. Coeficienții, definiți în cadrul clasei Term ca double, sunt testați pentru a fi identificat daca formatul lor este intradevăr de tip double. În cazul în care nu este (cazul cel mai probabil), valoarea double este converită în valoare integer, iar apoi adăugată în StringBuilder.
* private void checkResult(String res2) este o metodă folosită pentru testarea rezultatului. În cazul în care acesta este invalid, un mesaj corespunzător va fi afișat către utilizator.
* Private void oneOp și biOp sunt metode folosite pentru apelarea operațiilor, bazându-se pe enumerația IDOp.

**3.3.4. Pachetul mathOperations**

Pachetul mathOperations conține clasele ce implementeză operațiile matematice se ce efectuează pe polinoame.

* **Interfața Operation** este implementată de toate clasele. Aceasta conține metode generale ce ajută la efectuarea tuturor operațiilor matematice pe polinoame.

*default void sortPoly(ArrayList<Term> resultPoly, int size)* este o metodă ce sortează termenii polinomului dupa putere, de la cel mai mare la cel mai mic

*default void opPoly(ArrayList<Term> resultPoly, int size)* este o metodă ce compactează un polinom. Aceasta modelează polinomul astfel încât să fie un singur termen pentru fiecare putere

*default void mergePoly(ArrayList<Term> firstPoly, ArrayList<Term> secondPoly, ArrayList<Term> resultPoly)* este o metodă ce comasează două polinoame

*default void merge(ArrayList<Term> poly, ArrayList<Term> resultPoly)* este o metodă ce adaugă conținutul unui Polinom poly într-un Polinom resultPoly.

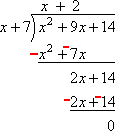
* **Clasa Sum** performează operația de adunare. Aceasta este facută prin apelul următaorelor metode: mergePoly(), sortPoly() și opPoly().
* **Clasa Difference** performează operația de scădere. Aceasta este făcută prin apelul metodei changeSign(), ce schimbă semnul coeficienților termenilor și prin apelarea metodelor mergePoly(), sortPoly() și opPoly(). Am ales implementarea folosind metodele interfeței și nu apelând după schimbarea semnului clasa Sum, pentru a nu creea dependențe între clasele de operații.
* **Clasa Multiplication** performează operația de înmulțire. Aceasta este făcură prin apelarea metodei multiply(), ce înmulțește fiecare termen al primului polinom cu fiecare termen al celui de-al doilea polinom, iar apoi prin apelarea metodelor implementate în interfață, sortPoly() și opPoly().
* **Clasa Division** performează operația de împărțire pe două polinoame. Aceasta este făcută prin aplicarea metodelor sortPoly() și opPoly() pe fiecare polinom, iar apoi prin apelarea metodei divide(). La final, rezultatul este aranjat prin apelarea metodelor sortPoly() și opPoly() pe polinomul rezultat. Implementarea algoritmului din metoda divide() va fi dezvoltată în cele ce urmează.
* **Clasa Derive** performează operația de derivare pe un polinom. Aceasta este făcută prin aplicarea metodelor sortPoly() și opPoly() pe polinomul transmis, iar apoi prin apelarea metodei deriveOp(), ce derivează fiecare termen al polinomului bazat pe principiile simple ale derivării.
* **Clasa Integral** este asemănătoare clasei Derive, cu diferență ca se apelează metoda integralOp pentru integrarea fiecărui termen al polinomului.

**3.4. Algoritmi**

Așa cum am prezentat și anterior, metodele a căror implementare a fost cea mai deficilă sunt deriveOp și pasteString. Voi mai detalia algoritmii folosiți în metodele sortPoly() și opPoly(), deoarece au fost metodele a căror apelare a fost cea mai des întâlnită.

3.4.1 *private void divide(ArrayList<Term> firstPoly, ArrayList<Term> secondPoly, ArrayList<Term> resultPoly)*

În cadrul acestei metode se produce împărțirea a două polinoame. Complexitatea acestui algoritm a fost dată de dorința mea de a avea un algoritm care să performeze împărțire exactă, respectiv să nu returneze restul împărțirii. Pentru aceasta, am implementat algoritmul matematic de Polynomial Long Division.



Un exemplu de funționalitate al acestui algoritm poate fi vazut în imaginea de mai sus. Implementarea algoritmului este în felul urmator:

Atâta timp cât condițiile sunt indeplinite, operează:

1) Termen al rezultatului este egal cu cel mai mare termen al împărțitorului, împărțit la cel mai mare termen al deîmpărțitului.

2) Se înmulțește ultimul termen al rezultatului obținut cu deîmpărțitul

3) Se scade din îmăpărțitor rezultatul înmulțirii precedente

4) Rezultatul scăderii devine noul Împarțitor

3.4.2 *private void parseString(String str, ArrayList<Term> poly)*

În cadrul acestei metode se performează conversia din format String în termeni ai polinomului ArrayList prin folosirea RegEx. Voi explica cele două formule Regular Expression pe care le-am folosit.

String[] output = str.split("(?=[-+])");

Această formulă împarte String-ul în substring-uri delimitate de semnele de anumare și scădere. Astfel, fiecare substring este un termen matematic al unul polinom.

String theRegex = "([+-]?[0-9]\*)x(\\^([0-9]+))?|([+-]?[0-9]+)";

Această formulă creează 3 grupuri, 2 dintre ele fiind esențiale. În grupul cu numărul 1, sunt puse elementele aflete între primele paranteze rotunde. Acestea corespund cu coeficientul termenului. Obersăm din expresie că putem avea un semn de plus sau minus, urmat de 1 sau mai multe cifre. În grupul cu numărul 3 sunt testate două cazuri, cel general, în care avem exponentul puterii și cazul în care avem termenul liber al polinomului.

3.4.3. default void sortPoly(ArrayList<Term> resultPoly, int size)

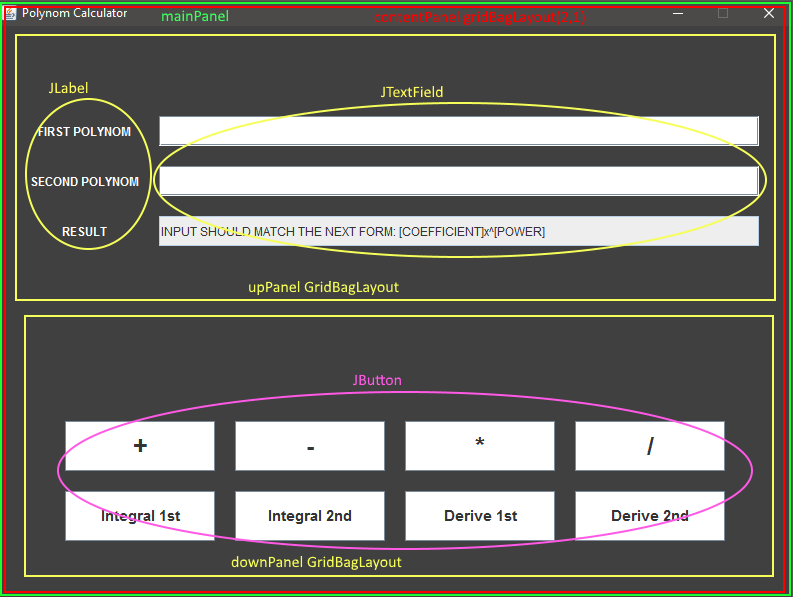
Algoritmul aflat în această metodă este, practic algoritmul de sortare QuickSort.

3.4.4. default void opPoly(ArrayList<Term> resultPoly, int size)

Rolul acestui algoritm este de a comasa termenii cu aceeași putere ai unui polinom. Acest lucru este făcut prin două bucle de tip for. Trebuie specificat că algoritmul este eficient doar pe o listă sortată. Primul for adună termenii cu peteri agale în primul termen cu acea putere găsit. Al doilea for elinimă toți termenii cu puteri egale găsiți cu excepția celui în care s-a făcut adunare în for-ul precedent.

**3.5. Interfața grafică**

Așa cum am detaliat și anterior, GUI este bazat pe implementare prin gridBagLayout. Designe-ul detaliat poate fi vazut in urmatoarea imagine :



4. Implementare și testare

Implementarea programului a fost făcută în totalitate în Eclipse. Principiile pe care m-am bazat au fost principiile de bază ale OOP, încercând să le respect cât mai mult. Nu am luat în calcul utilizarea memoriei sau viteza de executare a algoritmilor alesi, respectiv nu am considerat ca acestea sunt obiective importante în cadrul proiectului. Principalul obiectiv ales a fost realizarea unui program funcțional, bazat pe principiile OOP.

Testarea programului a fost făcută doar în cadrul Eclipse. Cu privire la modul de funționare, programul funționează perfect pe toate operațiile cu exceptia cazului în care lucrăm cu polinomul 0. Am conștientizat problema, iar în cazul necesității ca acesta sa fie rezolvată ar fi posibil. Această eroare se datorează lipsei de experientă în folosirea RegEx. Atata timp cât utilizatorul introduce polinoame sub forma cerută, în afara cazului mai sus menționat, toate operațiile returnează rezultatul corect. În cazul în care user-ul nu introduce polinoame, atunci programul nu se oprește, ci ii transmite utilizatorului mesajul că nu a introdus datele corect.

5. Rezultat

Rezultatul este o aplicație ce operează pe unul sau două polinoame, putând să efectueze operațiile de bază.

6. Concluzie

Acest proiect a reprezentat, în primul rând, o reinițiere în programarea OOP. Având puțină experiență anterioară, practic proiectul a însemnat reamintirea și consilidarea cunoștiintelor acumulate până acum. Un lucru care mi s-a părut foarte util, a fost prima întâlnire cu Regular Expressions. Consider că ar fi fost de mare ajutor să fie explicate într-un laborator. Cu siguranță, le voi mai folosi pe viitor.

7. Bibliografie

a) <http://stackoverflow.com/>

b) <http://www.tutorialspoint.com/>

c) <http://youtube.com/> - pentru tutoriale