**Documentatie-Tema 1**

**Procesare polinoame**

***Calacean Ionut Eugen***

***Facultatea de automatica si calculatoare***

***Grupa 30229***

***Indrumator laborator:Pop Cristina***

**Cuprins**

1. **Obiectivul temei.**
2. **Analiza problemei, modelare, scenarii, cazuri de utilizare.**
3. **Proiectare(decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator).**
4. **Implementare.**
5. **Rezultate.**
6. **Concluzii.**
7. **Bibliografie.**

**1.Obiectivul temei**

Polinoamele sunt construite din [termeni](https://ro.wikipedia.org/w/index.php?title=Termen_(matematic%C4%83)&action=edit&redlink=1) numiți [monoame](https://ro.wikipedia.org/w/index.php?title=Monom&action=edit&redlink=1), care sunt alcătuite dintr-o constantă (numită [coeficient](https://ro.wikipedia.org/wiki/Coeficient)) înmulțită cu una sau mai multe [variabile](https://ro.wikipedia.org/wiki/Variabil%C4%83). Fiecare variabilă poate avea un [exponent](https://ro.wikipedia.org/w/index.php?title=Exponent&action=edit&redlink=1) constant întreg pozitiv. Exponentul unei variabile dintr-un monom este egal cu [gradul](https://ro.wikipedia.org/w/index.php?title=Gradul_unui_polinom&action=edit&redlink=1) acelei variabile în acel monom. Pentru că x = x 1 {\displaystyle x=x^{1}} x=x^1, gradul unei variabile fără exponent este unu. Un monom fără variabile se numește monom constant, sau doar constantă. Gradul unui termen constant este 0. Coeficientul unui monom poate fi orice număr, inclusiv fracții, numere iraționale sau negative. Un polinom construit cu o singură variabilă se numește univariat.

Cerinta principala a temei este “Propuneti, proiectati si implementati un sistem de procesare a polinoamelor de o singura variabila cu coeficienti intregi ”, astfel obiectivul principal al temei este dezvoltarea unei aplicatii capabile sa efectueza operatiile principale pe polinoamele descrise in primul paragraf, cu observatia ca se vor trata polinoamele de o singura variabila cu coeficienti intregi.

Obiective secundare:

* Tratarea cazurilor de utilizare, prin diagrame use-case, aici prezentandu-se scenarii de utilizare a viitoarei aplicatii(capitol 2);
* Crearea operatiilor pe monom: adunare, scadere, inmultire, impartire, derivare si integrare(capitol 3);
* Extinderea operatiilor la nilvel de polinom(capitol 3);
* Crearea unei interfete utilizator capabila sa primeasca input, sa primeasca alegerea unei comenzi din cele 6 posibile si sa ofere un output corect si frumos afisat(capitol 4);
* Crearea unui Controller care sa lege efectiv interfata de model, adica de monom si polinom(capitol 3);
* Scenarii pentru testare si rezultatele acestora prezentate(capitol 5);
* Prezentare concluzii, posibilitati de dezvoltare ulterioara si cunostintele deprinse in urma proiectului(capitol 6).

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

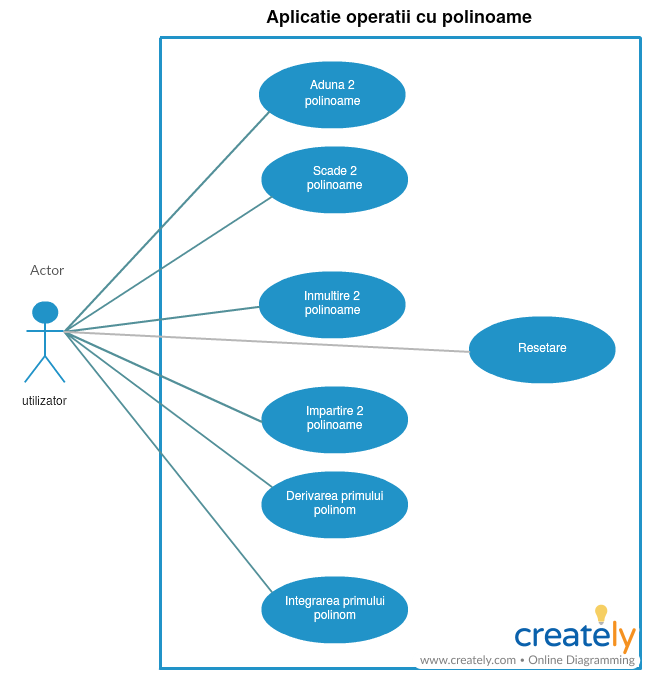
Analiza problemei ne imparte la o prima vedere sarcinile pe 6 ramuri definite de cele 6 operatii principale.Principiile programarii orientate pe obiecte ne permit sa vizualizam 2 clase importante, clasa monom si clasa polinom, pe care va trebui sa implementam operatiile si sa le legam pentru a obtine rezultatul dorit.

Pentru a lucra cu obiectele noastre, care sunt la baza monomul si polinomul avem nevoie de metodele: adunare, scadere, inmultire, impartire, derivare si integrare, implementate in ambele clase.

Din punct de vedere al interfetei, aveasta va permite utilizatorului sa introduca 2 polinoame( minim 1 polinom in cazul in care se doresc operatiile de derivare sau integrare), fapt care duce automat la necesitatea implementarii unei metode de conversie a string-ului introdus intr-un obiect de tip polinom.

Tot din procesul de modelare face parte si formatarea output-ului, astfel incat formatul acestuia sa semene cu formatul introdus de utilizator. Posibilitatea de introducere a unui string eronat, care sa nu respecte forma unui polinom nu trebuie scapata din vedere de catre programator, astfel o sa se trateze toate erorile de input.

In continuare voi prezenta scenarii si cazuri de utilizare sub forma de diagrame use-case:



* **Use case: 1 din cele 6 operatii**
* **Primary actor:utilizatorul**
* **Main succes scenario:**

1. Utilizatorul porneste aplicatia
2. Utilizatorul introduce in cele 2 spatii destinate polinoamele, sub forma de text
3. Utilizatorul selecteaza prin apasarea unui buton corespunzator unei operatii din cele 6
4. Aplicatia verifica daca polinoamele sunt introduse corect si daca operatia selectata se poate efectua
5. Aplicatia afiseaza rezultatul in spatiul destinat acestuia
6. Utilizatorul poate introduce alte date peste cele introduse sau poate selecta resetarea aplicatiei
7. Utilizatorul preia rezultatul furnizat de aplicatie

* **Use case: 1 din cele 6 operatii**
* **Primary actor:utilizatorul**
* **Alternative Sequences:**

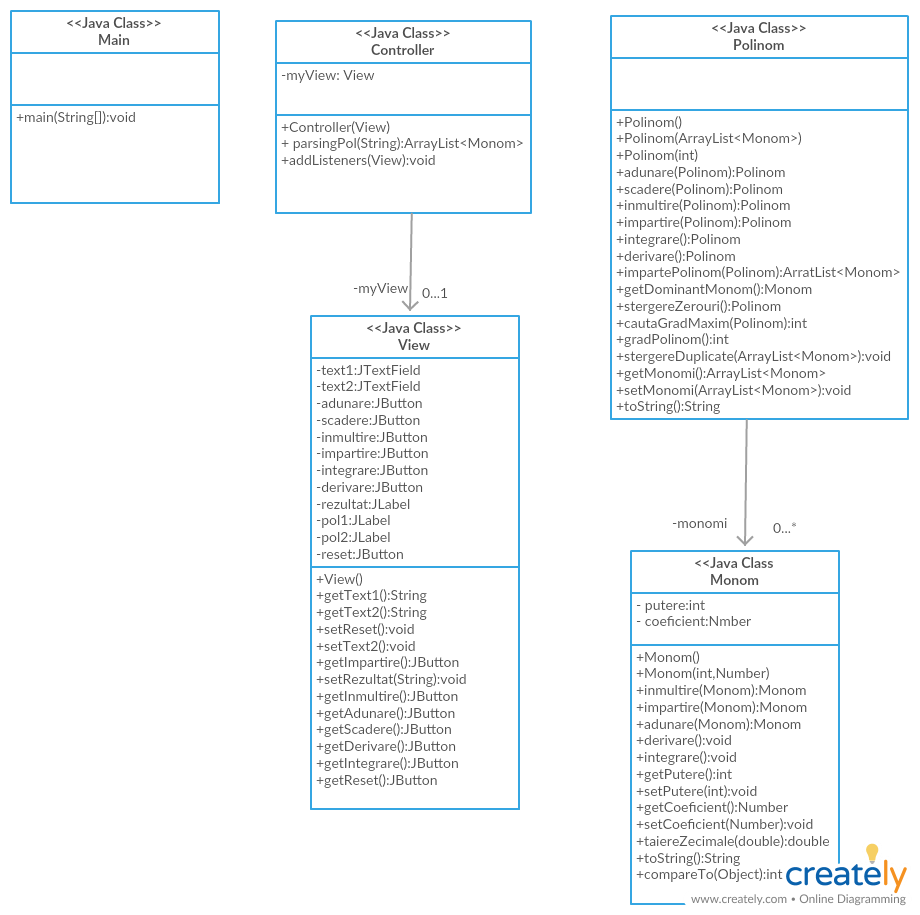
1. Polinoamele sunt introduse incorect

* Se detecteaza litere sau simboluri in afara celor permise, astfel se afiseaza un mesaj de tip warning
* Se apasa tasta OK si se reintroduc polinoamele

1. Operatia este incorecta considerand polinoamene

* Al doilea polinom este 0
* Se afiseaza un mesaj de eroare si se poate reveni la introducerea polinoamelor pentru corectarea erorii

**3.Proiectare**



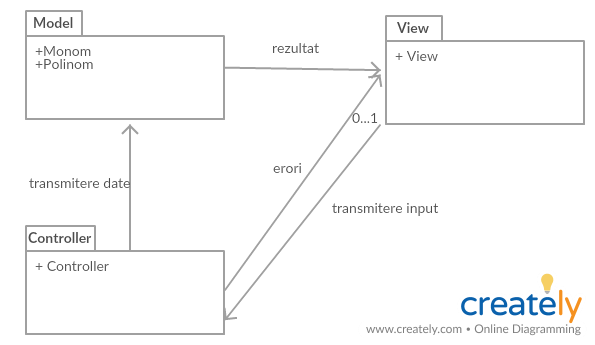
Clase folosite:

1. Clasa **Main**: - Cu ajutorul ei se leaga componentele si se ruleaza efectiv programul, aici fiind implementata metoda main;
2. Clasa **Monom**:- este clasa de baza a aplicatiei, contine coeficientul(intreg sau real) si puterea(de tip intreg), aici se definesc si operatiile de baza, intre monomii polinoamelor.In aceasta clasa se defineste si o conversie la string si o metoda de sortare, clasa implementand **interfata** Comparable;
3. Clasa **Polinom**:- aici se instantiaza polinoamele ca lista de monomi, sunt implementate operatiile, cat si unele metode precum cele de stergere a zerourilor, de eliminare a duplicatelor, de identificare a gradului, de conversie in string, ca un rezultat identic ca format cu inputul utilizatorului;
4. Clasa **Controller**:-in aceasta clasa se implementeaza ascultatorii, datele din interfata sunt preluate si transemise inspre modelul reprezentat de clasele monom si polinom, tot aici se realizeaza si transformarea string-urilor din input in polinoame efective;
5. Clasa **View**:-in cadrul acestei clase se construieste intefrata grafica si se ofera prin metode acces butoanelor si campurilor de tip text pentru a comunica cu Controllerul si cu Modelul;

Ca alte tipuri de date, in clasa Polinom se folosesc colectiile, pentru a pastra lista de monomi si a putea opera asupra acestora cu o mai mare usurinta, avand in vedere metodele puse la dispozitie si existenta clasei Iterator care permite parcurgerea si modificarea colectiei noastre.

Astfel, se observa strucutrarea proiectului in 3 mari pachete: Model, View si Controller, care comunica intre ele pentru a furniza rezultatul utilizatorului.

In continuare se va prezenta diagrama de pachete:



View-ul preia inputul de la utilizator si il transmite Controllerului, care prelucreaza si il transforma in date compatibile cu Modelul care prelucreaza datele si transmite mai departe rezultatul spre View. Astfel, Controllerul depinde de View printr-o relatie de agregare, depinde si de model prin rezultat. View-ul depinde de Controller, modificandu-se in cazul in care exista erori depistate de Controller, iar Modelul depinde prin intermediul Controller-ului de View.

**4.Implementare**

* **Clasa Monom.**

Ca variabile contine puterea si coeficientul, iar clasa implementeaza interfata Comparable<Object> pentru a putea sorta frumos monomii in outputul pe care il oferim utilizatorului, acest lucru se face prin suprascrierea metodei compareTo(Object arg0).

Pe langa implementarea operatiilor de baza pe monomi care nu necesita o expicatie suplimentara, o metoda importanta din aceasta clasa este transformarea monomului in string in functie de coeficientul care poate sa fie intreg sau real. Am prezentat in continuare metoda:

*@Override*

public String toString() {

int coeficientIntreg;

double coeficientZecimal;

if (coeficient.doubleValue() != coeficient.intValue()) {

coeficientZecimal = this.taiereZecimale(coeficient.doubleValue());

if (putere != 0) {

if (coeficientZecimal > 0) {

return "+" + coeficientZecimal + "X^" + putere;

}

return coeficientZecimal + "X^" + putere;

}

if (coeficientZecimal > 0)

return "+" + coeficientZecimal;

return coeficientZecimal + "";

}

else {

coeficientIntreg = coeficient.intValue();

if (putere != 0) {

if (coeficientIntreg > 0)

return "+" + coeficientIntreg + "X^" + putere;

return coeficientIntreg + "X^" + putere;

}

if (coeficientIntreg > 0)

return "+" + coeficientIntreg;

return coeficientIntreg + "";

}

}

* **Clasa Polinom.**

Clasa contine o colectie de tip ArrayList<Monom> care permite stocarea monomilor componenti. Exista 3 constructori publici: unul fara parametrii care creeaza o lista goala in care se vor adauga ulteriori monomi, un constructor cu parametru ArrayList<Monom> care construieste un polinom direct pe baza monomilor constituenti si un constructor care primeste ca parametru un intreg care reprezinta gradul unui polinom. Acest al treilea constructor este unul mai special, in sensul ca ne creeaza un polinom de un grad dat cu toti coeficientii 0, pentru a creste maleabilitatea operatiilor pe care le dorim implementate.

In continuare se prezinta operatiile principale, unele operatii secundare asupra setului de date si bucati din cod pentru a intelege mai bine explicatiile aferente.

1. **Adunarea a doua polinoame**

Principiul adunarii se bazeaza pe insumarea coeficientilor monomilor cu grad egal. In implementarea aleasa, se cauta intai gradul polinoamelor si se alege cel mai mare, pe baza acestui grad se construieste un polinom rezultat, cu coeficientii 0, cu ajutorul constructorului special.

In polinomul rezultat se aduna pe rand cele 2 polinoame, adunand monomul cu un anumit grad din parcurgerea de tip ‘foreach’ la monomul cu acelasi grad din polinomul rezultat.

Principiul este prezentat in urmatoarele linii de cod:

public Polinom adunare(Polinom secondPol) {

int gradMaxim, gradCurent;

if (secondPol.gradPolinom() > this.gradPolinom())// am selectat gradul maxim

gradMaxim = secondPol.gradPolinom();

else

gradMaxim = this.gradPolinom();

Polinom newPol = new Polinom(gradMaxim + 1);

Monom monomCurent;

for (Monom list : monomi) { // se aduna polinomul curent la noul polinom

gradCurent = list.getPutere();

monomCurent = newPol.getMonomi().get(gradCurent).adunare(list);

newPol.getMonomi().set(gradCurent, monomCurent);

}

for (Monom list : secondPol.getMonomi()) { // se aduna al doilea polinom

gradCurent = list.getPutere();

monomCurent = newPol.getMonomi().get(gradCurent).adunare(list);

newPol.getMonomi().set(gradCurent, monomCurent);

}

newPol.stergereZerouri();

return newPol;

}

1. **Scaderea a doua polinoame**

Pentru scadere nu exista un algoritm special, fiind doar o adunare cu semn schimbat, astfel se iau toti monomii din cel de-al doilea polinom si se inmultesc coeficientii sai cu (-1).

Se apeleaza in continuare adunarea implementata la pasul anterior.

1. **Inmultirea a doua polinoame**

Principiul consta in inmultirea fiecarui monom din primul polinom cu fiecare monom din cel de-al doilea polinom, toate rezultatele inmultirilor se aduna in polinomul rezultat.

Dupa acest procedeu apelam metoda statica sort implementata in Collections, pentru a obtine monomii in ordinea descrescatoare a gradului.

Tot in interiorul metodei de inmultire se apeleaza 2 metode care vor fi prezentate ultrior, stergereDuplicate si stergereZerouri. Codul aferent este prezentat in continuare:

public Polinom inmultire(Polinom secondPol) {

ArrayList<Monom> newPol = new ArrayList<Monom>();

for (Monom mon1 : this.getMonomi()) {

for (Monom mon2 : secondPol.getMonomi()) {

newPol.add(mon1.inmultire(mon2));

}

}

Collections.*sort*(newPol);

stergereDuplicate(newPol);

Polinom rezultat = new Polinom(newPol);

rezultat.stergereZerouri();

return rezultat; }

Metoda stergereZerouri consta intr-o simpla parcurgere a polinomului si elimiare a monomilor care contin coeficientul 0, iar metoda stergereDuplicate itereaza prin lista de monomi, si daca exista 2 monomi cu grad identic, aduna coeficientii, si apoi sterge un monom duplicat. Aceste metode sunt utile pentru comprimarea rezultatului si obtinerea unui format identic cu cel al inputului cerut.

1. **Impartirea a doua polinoame**

Impartirea a doua polinoame este probabil cea mai complexa operatie din cele prezentate, astfel se face o transpunere matematica in cod a procedeului de impartire invatat in liceu.

Acest procedeu consta in existenta unui polinom rezultat si a unui polinom auxiliar. Astfel, se construieste o metoda secundara impartePolinom care imparte monomii dominanti din cele cele 2 polinoame(deimpartit si impartitor), cu acest rezultat se inmulteste impartitorul si se salveaza in polinomul auxiliar. Polinomul auxiliar obtinut se scade din deimpartit. Procedeul se repeta pana cand gradul deimpartitorului devine mai mic decat gradul impartitorului. Ce va ramane in primul polinom(deimpartitul) va constitui restul.

Constructia polinomului rezultat se face astfel treptat, de fiecare data se impart monomii dominanti si se aduna la acest polinom rezultatul.

Procedeul este prezentat simplificat in codul care urmeaza:

while (pol1.gradPolinom() >= secondPol.gradPolinom()) {

auxiliar = new Polinom(pol1.impartePolinom(secondPol));

rezultat = rezultat.adunare(auxiliar);

auxiliar = auxiliar.inmultire(secondPol);// cel mai semnificativ se inmultetste cu impartitorul

pol1 = pol1.scadere(auxiliar);// scadem din polinomul 1 pe cel auxiliar

}

1. **Derivarea unui polinom**

Pentru derivarea polinomului se ia primul polinom introdus de utilizator se parcurge colectia de monomi si se deriveaza pe rand. Suplimentar este necesara o apelare a metodei de stergere a zerourilor:

public Polinom derivare() {

Polinom pol = new Polinom(this.getMonomi());

for (Monom list : pol.getMonomi()) {

list.derivare();

}

pol.stergereZerouri();

return pol;

}

1. **Integrarea unui polinom**

Integrarea unui polinom este similara derivarii, doar ca aici nu mai este nevoie de apelarea metodei suplimentare:

public Polinom integrare() {

Polinom pol = new Polinom(this.getMonomi());

for (Monom list : pol.getMonomi()) {

list.integrare();

}

return pol;

}

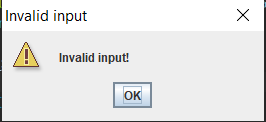
* **Clasa Controller**

Aceasta clasa contine un constructor prin care se leaga View-ul la ea si se adauga acestuia ascultatori. Avem doar doua metode in aceasta clasa cu o importanta primordiala, si anume:

public static ArrayList<Monom> parsingPol(String pol)

public void addListeners(final View myView)

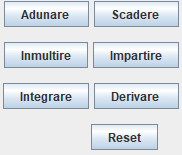
Prima metoda contine transformarea efectiva a inputului utilizatorului care este de tipul String, intr-o lista de monomi care ne va ajuta ulterior la construirea polinoamelor efective. Utilizatorul introduce 2 polinoame cu ajutorul caracterelor ‘x’, ‘X’, ‘^’ ,’+’, ‘-‘ si evident cifre. Tot aici se trateaza si cazul unui input invalid, moment in care o sa se afiseze un WARNING\_MESSAGE care sa avertizeze utilizatorul in privinta greselii.:



Trecerea de la forma de String la polinomul efectiv se face folosind Expresii Regulare, sau denumirea pe scurt **regex.** Principiul este urmatorul: o astfel de expresie defineste un anumit model pentru siruri de caractere. In continuare, pe langa clasa **Pattern** pusa la dispozitie de catre **Java** mai exista si clasa **Matcher.** Modelul de cautare poate fi un simplu caracter, un sir fix, sau o expresie complexa care contine caractere speciale descrise de model(la noi prezenta ‘^’ este un exemplu concludent).

Stringul regex folosit de mine:String regex = "(?:\\h\*)([-+]?\\d\*)(?:\\h\*)[xX](\\^(\\d+))?|(?:\\h\*)([-+]?\\d+)" . Acest string contine cazul in care avem +- cueficient si putere, doar ‘x’ sau doar coeficient. Se apeleaza metoda find din clasa Matcher si astfel se verifica in care caz ne aflam, pe baza acestuia se construieste pe rand cate un monom si se adauga rezultatului.

La fiecare selectare de operatie, cele 2 polinoame se construiesc iar, astfel evitam sa obtinem rezultate gresite si avem mereu 2 polinoame cu care putem lucra.

 Metoda de adaugare ascultatoare pentru butoane impune crearea de clase ActionListener() care implementeaza metoda actionPerformed() astfel se detecteaza operatia selectata de utilizator, se construiesc polinoamele si se aplica operatia respectiva, dupa care se pune rezultatul in interfata. Butoanele la care se adauga ascultatoare sunt urmatoarele:

* **Clasa View**

Aceasta clasa contine un JFrame pe baza caruia se construieste intreaga interfata:

JFrame frame = new JFrame("Polinoame");

frame.setSize(800, 600);

frame.setContentPane(finalPane);

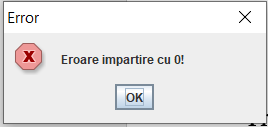
frame.setVisible(true);

frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

Ca mod de aranjare a componentelor pe fereastra, s-a folosit un layout de tip BoxLayout, asupra acestui tip de aranjare programatorul avand o mai mare putere de modelare. Ca o structura a interfetei utilizator, aceasta se imparte in 2 mari zone:

1. **Zona de input**- in care utilizatorul introduce cele 2 polinoame (sau unul in functie de operatia dorita) si selecteaza un buton de control (exita un buton pentru fiecare operatie dorita si pentru reset);
2. **Zona de outpui**- in aceasta zona utilizatorul are primeste rezultatul daca polinoamele introduse sunt corecte din punct de vedere al sitaxei si operatia dorita este posibila;

O eroare a utilizarii aplicatiei se va obtine in cazul in care utilizatorul doreste impartirea la 0 (care nu este posibila matematic), momentul in care utilizatorul apasa butonul impartire si al doilea polinom este ‘0’ aplicatia va detecta eroarea si va afisa un mesaj de tip ERROR\_MESSAGE:



Din punct de vedere al componentelor folosite, se va prezenta in continuare codul corespunzator si o imagine a interfetei. Componentele foliste sunt doar dintre tipurile JTextField, JButton, JLabel:

private JTextField text1 = new JTextField(30);

private JTextField text2 = new JTextField(30);

private JButton adunare = new JButton("Adunare ");

private JButton scadere = new JButton("Scadere ");

private JButton inmultire = new JButton("Inmultire");

private JButton impartire = new JButton("Impartire");

private JButton integrare = new JButton("Integrare");

private JButton derivare = new JButton("Derivare ");

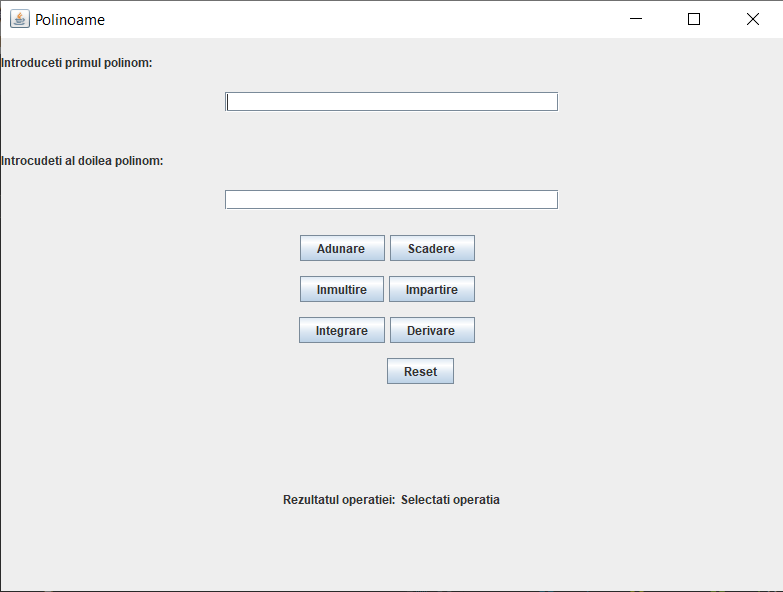
private JLabel rezultat = new JLabel("Selectati operatia");

private JLabel pol1 = new JLabel("Introduceti primul polinom:");

private JLabel pol2 = new JLabel("Introcudeti al doilea polinom:");

private JButton reset=new JButton("Reset");

Toate aceste componente sunt adaugate unor componente de tip JPanel, pentru a putea jongla cu ele si a le aseza unde dorim, intr-un final, dupa folosirea layout-urilor dorite, toate componentele se adauga Containerului finalPane care este tot de tipul JPanel. Imaginea finala a interferei este urmatoarea:



**5.Rezultate**

Majoritatea cazurilor testate sunt corecte din punct de vedere matematic iar maleabilitatea inputului de la utilizator este destul de mare, utilizatorul putand sa introduca ‘^1’ sau sa omita acest lucru, sa introduca spatii suplimentare.

Pot insa exista unele cazuri cand inputul este fortat in anumite feluri si outputul nu este cel asteptat.

In continuare se vor prezenta unele scenarii pentru testare implementate folosind biblioteca **org.junit:**

Polinom p1, p2, p3;

String pol1 = "x^4+3x^2+2x-1", pol2 = "x^2+2x+3";

*@Before*

public void setUp() throws Exception

{

p1 = new Polinom(Controller.*parsingPol*(pol1));

p2 = new Polinom(Controller.*parsingPol*(pol2));

}

*@Test*

public void adunare() throws Exception

{

p3 = p1.adunare(p2);

*assertEquals*("+1X^4+4X^2+4X^1+2", p3.toString());

}

*@Test*

public void scadere() throws Exception

{

p3 = p1.scadere(p2);

*assertEquals*("+1X^4+2X^2-4", p3.toString());

}

*@Test*

public void inmultire() throws Exception

{

p3 = p1.inmultire(p2);

*assertEquals*("+1X^6+2X^5+6X^4+8X^3+12X^2+4X^1-3", p3.toString());

}

*@Test*

public void impartire() throws Exception

{

p3 = p1.impartire(p2);

*assertEquals*("+1X^2-2X^1+4 Rest: -13", p3.toString() + " Rest: " + p1.toString());

}

*@Test*

public void integrare() throws Exception

{

p3 = p1.integrare();

*assertEquals*("+0.2X^5+1X^3+1X^2-1X^1", p3.toString());

}

*@Test*

public void derivare() throws Exception

{

p3 = p1.derivare();

*assertEquals*("+4X^3+6X^1+2", p3.toString());

}

Rezultatele acestor scenarii de test sunt cele asteptate, pentru aceasta se creeaza clasa TestRunner care ruleaza efectiv testele folosind **org.junit.runner.\*:**

Result result=JUnitCore.*runClasses*(PolinomTest.class);

for(Failure failure: result.getFailures())

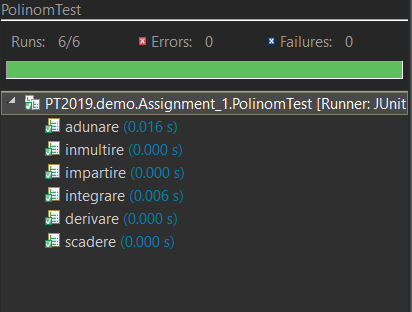
{

System.***out***.println(failure.toString());

}

System.***out***.println(result.wasSuccessful());

Imagine cu rezultatele dupa rulare:



**6.Concluzii**

Ca o concluzie la acest proiect, se observa importanta aplicarii conceptelor matematice in programare, pentru a obtine rezultate corecte. Cel mai important aspect din acest proiect, care m-a ajutat la dezvoltarea cunostintelor de programare este parsarea stringurilor. Puterea de modelare a programatorului prin folosirea Expresiilor regulare(regex) este o noutate pentru mine, iar maleabilitatea oferita de modelele suportate (pattern-urile de caractere recunoscute) face ca programatorul sa fie capabil de identificarea oricarui string cautat.

Din punct de vedere al dezvoltarilor ulterioare, se mai poate lucra la interfata cu utilizatorul, in sensul ca s-ar putea introduce un buton pentru a obtine un input exact similar cu un polinom in matematica(similar cu modelele de calculatoare stiintifice care permit introducerea puterii fara ‘^’). O alta idee de dezvoltare presupune implementarea unor metode care sa faca transformata Laplace a polinomului introdus.

Tot in cadrul ideilor de dezvoltare intra si posibilitatea de a introduce o functie de transfer si o anumita intrare(pe cele 2 TextField-uri pentru polinoame) si obtinerea unei iesiri pe baza datelor introduse.

Cea mai importanta cunostinta deprinsa este aceea de a intra pana la baza unei probleme si a rezolva cerintele( de exemplu se rezolva operatiile pe fiecare monom in parte) astfel la revenirea spre generalizarea problemei, cerintele sunt intr-un fel mai usoare.

**7.Bibliografie**

[1] <https://www.vogella.com/tutorials/JavaRegularExpressions/article.html>

[2] <https://www.mathsisfun.com/algebra/polynomials-division-long.html>

[3] <http://users.utcluj.ro/~crisb_pop/pt2018.html>

[4] <https://docs.oracle.com/javase/tutorial/uiswing/layout/layoutlist.html>