**DOCUMENTATIE TEMA 1**

**CALCULATOR POLINOAME**

**Nume prenume Man Larisa Silvana**

**Grupa 30224**

**Profesor Laborator Assist Pop Claudia**

Contents

[1. Cerinte Functionale 3](#_Toc477463628)

[2. Obiective 3](#_Toc477463629)

[2.1. Obiectiv Principal: 3](#_Toc477463630)

[2.2. Obective Secundare: 3](#_Toc477463631)

[3. Analiza Problemei 4](#_Toc477463632)

[4. Proiectare 5](#_Toc477463633)

[4.1. Structuri de date 5](#_Toc477463634)

[4.2. Diagrama de clase 6](#_Toc477463635)

[4.3. Algoritmi 6](#_Toc477463636)

[5. Implementare 8](#_Toc477463637)

[Testare 9](#_Toc477463638)

[6. Concluzii si Dezvoltari Ulterioare 11](#_Toc477463639)

[7. Bibliografie 11](#_Toc477463640)

# Cerinte Functionale

Dezvoltati un calculator de polinoame care sa perimita urmatoarele operatii:

* Citirea unui polinom de la tastatura sub forma 3X^2+2X^3-X+2
* Adunare, Scadere, Inmultire, Impartire, Derivare, Integrare

# Obiective

## Obiectiv Principal:

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

Obiectivul temei este de a implementa si a testa algoritmi pentru efectuarea operatiilor specifice asupra polinoamelor. Aceste operatii trebuie implementate folosind limbajul de programare Java si conceptele de programare orientata obiect. Proiectul este realizat in mediul de dezvoltare Eclipse folosing interfata grafica, navigarea prin aplicatie se face cu mouseul prin apasarea butoanelor si cu tastatura pentru a introduce polinoamele.

Operatiile implementate asupra polinoamelor sunt: *adunarea*, *scaderea*, *inmultirea*, *impartirea*, *derivarea* si *integrarea.* Pentru implementarea acestor operatii s-au folosit algoritmii matematici de procesare a polinoamelor.

Ideea este de a vedea un polinom ca o suma de monoame. Operatiile sunt facute pe obiecte de tip polinom,dar toate polinoamele sunt o suma de obiecte de tip Monom. Polinoamele au coeficienti intregi, dar pentru a usura munca in implementarea operatiilor de impartire si integrare, am ales ca acestia sa fie declarati ca fiind reali (double). Interfata grafica este alcatuita dintr-un JFrame care primeste comenzi de la mouse si tastatura.

## Obective Secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Dezvoltarea de use case-uri si scenarii | Este necesara crearea unei diagrame USE CASE pentru a avea o idee clara asupra modului in care actorii vor interactiona cu sistemul informatic. Ofera o imagine generala asupra sistemului. | 3 |
| Alegerea structurilor de date | Vom alege structura de date cea mai potrivita pentru modelul nostru, in acest caz ArrayList, deoarece valorile sunt accesate folosind o pozitie întreagă. | 4 |
| Impartirea pe clase | Clasele principale folosite sunt Monom, Polynom si Operation. Acestea au fost create gandindu-ne la cea mai usoara modalitate de a le lega intre ele si accesa. | 4 |
| Dezvoltarea algoritmilor | Clasa Operation contine toti algoritmii folositi in implementarea operatiilor folosite de calculatorul nostru. | 4 |
| Implementarea solutiei | Pentru implementare am folosit mediul de dezvoltare Eclipse si am sortat clasele in diferite pachete, in functie de utilizarea lor. | 5 |
| Testare | Am folosit clasa Junit pentru a testa produsul. | 6 |

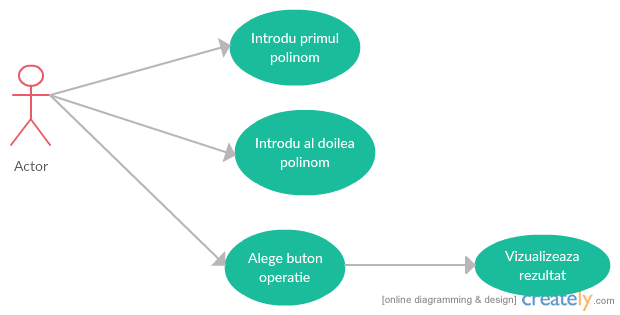
# Analiza Problemei

Implementarea acestui proiect necesita atat cunostinte de matematica cat si cunostinte de programare orientate pe obiect, dobandite in semestrul anterior.

In acest proiect lucram trebuie sa lucram cu polinoame de numere intregi, dar in cazul efectuarii operatiilor de impartire si integrare este posibil ca coeficientii polinomului rezultant sa aiba coeficienti reali. De aceea. Vom folosi coeficienti reali in implementare.

Satisfacand conditiile amintite mai sus, am ca reprezentarea polinoamelor sa fie ca o lista de monoame, fiecare monom avand un grad si un coeficient.

Introducerea polinoamelor se face cu ajutorul interfetei grafice. Utilizatorul va introduce doua polinoame de tipul 2X^3+6X^2+5X^0, iar apoi cu ajutorul butoanelor va alege operatia dorita pe polinoame.



# Proiectare

## Structuri de date

O listǎ este o colecţie **ordonatǎ**. Listele pot conţine elemente **duplicate**. Pe langǎ operaţiile moştenite de la Collection, interfaţa List defineşte urmǎtoarele operaţii:

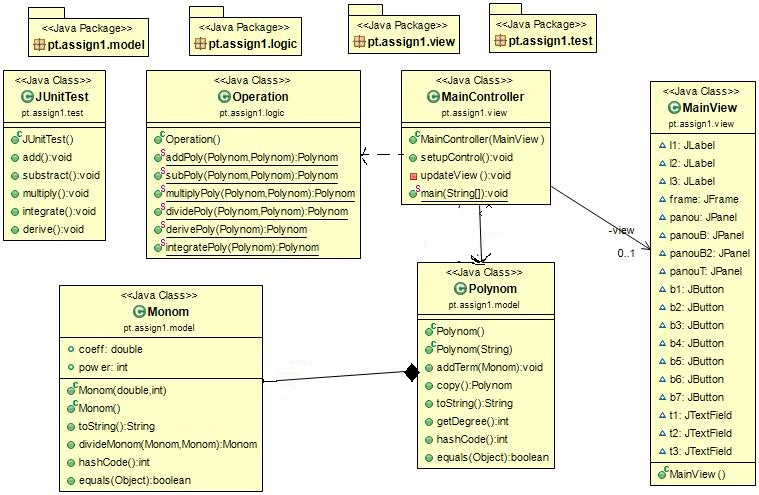
* T get(int index) - întoarce elementul de la poziţia index
* T set(int index, T element) - modificǎ elementul de la poziţia index
* void add(int index, T element) - adaugǎ un element la poziţia index
* T remove(int index) - şterge elementul de la poziţia index

Alǎturi de List, este definitǎ interfaţa ListIterator, ce extinde interfaţa Iterator cu metode de parcurgere în ordine inversǎ.  
List posedǎ implementarea ArrayList - implementare sub formǎ de vector. Accesul la elemente se face în timp constant: O(1).

Am ales ca structura de date ArrayList, deoarece valorile sunt accesate folosind o pozitie întreagă.

Polinomul este este un ArrayList de monoame, o lista sortata, primul monom fiind cel cu gradul mai mare.

## Diagrama de clase



Proiectul meu consta in 4 pachete: pt.assign1.logic, pt.assign1.model, pt.assign1.test, pt.assign1.view. Pachetul “logic” contine informatiile din pachetul “model”. In pachetul “model” gasim clasele Monom si Polynom, care reprezinta modelarea proiectului. In pachetul “logic” gasim clasa Operation, care consta in implementarea algoritmilor pentru realizarea operatiilor pe polinoame. Pachetul “view” contine doua clase esentiale : MainController care este capul sistemului si clasa MainView, care implementeaza intrfata grafica. In pachetul “test” se realizeaza testarea cu ajutorul clasei Junit.

## Algoritmi

In acest subcapitol se vor descrie algoritmii folositi in realizarea operatiilor pe polinoame. Cand am implementat metodele respective, am considerat relatiile matematice pentru operatii astfel incat sa obtinem un rezultat corect, apoi am incercat transpunerea relatiilor in cod, rezultand operatiile finale.

**Metoda de adunare AddPoly** are ca parametri doua polinoame si returneaza un polinom rezultat din suma celor doua. Metoda va fi apelata astfel:

result = **Operation**.*addPoly*(p1, p2); //unde result, p1, p2 sunt polinoame

Metoda consta in crearea a unor copii ale polinoamelor date ca parametri, pentru a ne asigra ca polinoamele initiale nu se vor modifica de-a lungul procesului. Copia celui de-al doilea polinom de va parcurge si se va aduna fiecare monom la prima copie a polinomului cu ajutorul metodei addTerm implementata in Polynom.

**Metoda de scadere subPoly** are ca parametri doua polinoame si returneaza un polinom rezultat din suma celor doua. Metoda va fi apelata astfel:

result = **Operation**.*subPoly*(p1, p2); //unde result, p1, p2 sunt polinoame

De asemenea se lucreaza cu copii ale polinoamelor si se va folosi aceeasi metoda de a aduna, in plus inmultindu-se coeficientul cu -1.

**Metoda de inmultire multiplyPoly** are ca parametri doua polinoame si returneaza un polinom rezultat din suma celor doua. Metoda va fi apelata astfel:

result = **Operation**.*multiplyPoly*(p1, p2); //unde result, p1, p2 sunt polinoame

In doua cicluri for se vor parcurge polinoamele si se vor inmulti monoamele intre ele.

**Metoda de impartire dividePoly** are ca parametri doua polinoame si se va afisa catul lor. Rezultaul impartii va avea un polinom **cat** si unul **rest.** Aceasta operatie se poate efectua doar in cazul in care primul polinom (deimpartitul) este de grad mai mare sau egal cu gradul impartitorului. La inceput se verifica aceasta conditie, si daca nu se indeplineste, executia operatiei se intrerupe. La inceput deimpartitul ia valoarea primului polinom, si se executa un ciclu de operatii pana cand conditia referitoare la gradele polinoamelor este adevarata. Se calculeaza raportul monoamelor de grad maxim a celor 2 polinoame. Rezultatul se adauga la cat. Apoi, se calculeaza restul: impartitorul se inmulteste cu monomul obtinut mai inainte. Scazand din deimpartit acest polinom se obtine restul. Acest rest devine noul deimpartit si procedeul se repeata.

Metoda va fi apelata astfel:

result = **Operation**.*dividePoly*(p1, p2); //unde result, p1, p2 sunt polinoame

**Metoda de derivare derivePoly** are ca parametru un polinom. Va fi luat in considerare primul polinom introdus de utilizator. Se va parcurge si monoamele vor fi modificate dupa regula. Metoda va fi apelata astfel:

result =**Operation**.*derivePoly*(p1); //p1, result polinoame

**Metoda de integrare integratePoly** are aceleasi proprietai ca si cea de derivare si se apeleaza astfel:

result = **Operation**.integrate*Poly*(p1);

# Implementare

**Clasa Monom** este definita de un coeficient de tip double si puterea de tip integer. Contine un constructor; metoda toString() care ajuta la afisarea monoamelor dupa diferite reguli; metoda divideMonom care imparte doua monoame intre ele si returneaza tot un monom; metodele HashCode si equals de care avem nevoie pentru a folosi assert in testarea unitara.

**Clasa Polynom** contine o lista de monoame, constructorul in care facem si preluarea polinomului de la intrare si spargerea lui cu ajutorul metodei Regex. Metoda addTerm() adauga monoame in lista ce va constitui in final polinomul nostru. Metoda copy() creeaza o copie a unui polinom. Aceasta metoda este folosita in operatii pentru ca polinomul intial sa ramana intact. Metoda toString() este folosita pentru afisare. Daca lista este goala se va afisa mesajul de eroare ERR. Metoda getDegree() gaseste gradul polinomului, adica cea mai mare putere a monoamelor. In final apar din nou metodele HashCode si equals, folosite pentru testarea unitara.

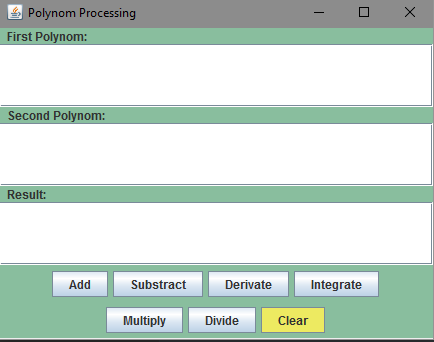
Clasele prezentate mai sus fac parte din pachetul “model” si reprezinta baza proiectului nostru.

**Clasa Operation** face parte din pachetul “logic”. Aici sunt implementati algoritmii folositi de calculator. Gasim cele 6 metode statice: addPoly, subPoly, multiplyPoly, dividePoly, derivePoly si integratePoly. Acestea au fost descrise in sectiunea 4.3.

Calculatorul este structurat in stilul MVC (Model- View- Controller). Modelul l-am vazut in pachetul “model”, iar clasele MainView si MainController le vom gasi in pachetul “view”.

**Clasa MainView** contine un frame in care vom gasi interfata grafica cu utilizatorul, 3 label-uri, 4 panouri, 3 textfield-uri si 7 butoane. Acestea sunt combinate si legate intre ele astfel incat sa ofere o interfata grafica placuta utilizatorului si usor de utilizat. Pachetul utilizat pentru crearea a GUI este Java Swing. In cele 4 JPanel-uri sunt aranjate celelalte elemente cat mai structurat. JLabel-urile arata ce date trebuie introduse in JTextField-uri. Cele 7 JButtons sunt pentru a efectua operatia de adunare, scadere, inmultire, impartire, integrare, derivare si un buton de CLEAR care curate JTextField-urile.

**Clasa MainController** este cea care controleaza interfata grafica, cu ajutorul butoanelor, Contine o metoda setupControl(), in care fiecarui buton ii este atribuit un ActionListener ce arata functionalitatea lui. Metoda updateView() este cea care atribuie o noua valoare rezultatului, in functie de butonul apasat. Tot aici gasim functia main() – capul proiectului, care creeaza un nou MainView si MainController.



# Testare

Am creat un nou pachet numit “test”, care contine o clasa numita JUnitTest. In aceasta clasa regasim metodele add, substract, multiply, derive si integrate, care testeaza fiecare operatie in parte: adunare, scadere, inmultire, derivare, integrare. Am ales intamplator polinoamele care vor fi testate cu ajutorul JUnit-ului.

De exemplu, in metoda add() am creat doua polinoame: p1=x^2 + x -1 si p2=x^2 -6x +8, carora le-am aplicat metoda addPoly(), implementata in clasa Operation. Rezultatul acesta a fost numit ca fiind un polinom “actual”. Tot aici am creat un polinom “expected”, adica rezultatul pe care il asteptam: 2x^2 -5x +7. Cu ajutorul functiei assertEquals, apartinand clasei Assert, am comparat cele doua rezultate: expected si actual. Daca ele sunt egale, atunci testarea se va afectua cu success.

In metoda multiply(), se aplica acelasi algoritm si folosim aceleasi polinoame ca si pentru adunare: p1=x^2 + x -1 si p2=x^2 -6x +8. Acestora le aplicam metoda multiplyPoly() si rezultatul il vom pune din nou intr-un polinom actual. Apoi cream polinomul “expected”: -6x^3 +x^4 +8x^2 -6x^2 +x^3 +8x +6x -x^2 -8=-5x^3 +x^4 +x^2 +14x -8 si il comparam din nou cu cel “actual”, cu ajutorul functiei assertEquals.

In metoda integrate(), cream un polinom 12x^3+ 3x^2 + x, caruia ii aplicam functia integratePoly din clasa Operation si punem rezultatul intr-un nou polinom. La fel, vom crea polinomul asteptat 3x^4 +x^3 +0.5x^2, care va fi comparat cu cel de mai sus.

***@Test***

**public** **void** **integrate**) **throws** **Exception** {

// 12x^3+ 3x^2 + x

**Monom** **term11** = **new** Monom(12,3);

**Monom** **term12** = **new** Monom(3,2);

**Monom** **term13** = **new** Monom(1,1);

**Polynom** **p1**=**new** Polynom();

p1.addTerm(term11);

p1.addTerm(term12);

p1.addTerm(term13);

**Polynom** **actual** = **Operation**.*integratePoly*(p1);

//3x^4 +x^3 +0.5x^2

**Monom** **expectedTerm1** = **new** Monom(3,4);

**Monom** **expectedTerm2** = **new** Monom(1,3);

**Monom** **expectedTerm3** = **new** Monom(0.5,2);

**Polynom** **expected** = **new** Polynom();

expected.addTerm(expectedTerm1);

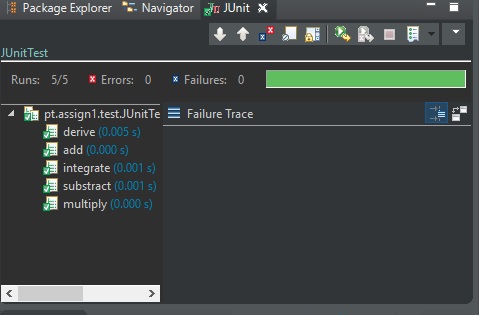
expected.addTerm(expectedTerm2);

expected.addTerm(expectedTerm3);

**Assert**.*assertEquals*(expected, actual);

}

Daca toate testele decurg corect, se va deschide o fereastra JUnit, unde vom vedea rezultatul:



# Concluzii si Dezvoltari Ulterioare

Rezultatele obtinute in urma prelucrarilor polinoamelor date de utilizator pot fi observate in interfata grafica si verificate prin realizarea calculelor in mod paralel, pe hartie. Astfel, este usor de remarcat faptul ca solutiile obtinute in urma apelarii metodelor din clasa Polynom sunt cat se poate de corecte, lucru ce atesta coerenta si eficienta algoritmilor implementati pentru realizarea acestora.

Ca o concluzie, acest proiect m-a ajutat sa imi consolidez cunostintele de programare orientata pe obiect dobandite in primul semestru si sa imi organizez munca in baza paradigmelor POO. Totodata, felul in care am structurat proiectul si am creat clasele si metodele m-a ajutat sa eficientizez codul din punct de vedere al lungimii si al usurintei intelegerii acestuia, lucruri vitale in cazul programelor complexe care necesita mai multe clase sau care presupun un numar mai mare de programatori.

Nu in ultimul rand, marele castig in implementarea acestui proiect a fost faptul ca am invatat sa realizez o interfata grafica, lucru ce imi dezvolta posibilitatile de implementare a noi proiecte in viitor.

In cadrul dezvoltarilor ulterioare se pot aminti urmatoarele: implementarea de noi operatii pentru polinoame, cum ar fi aflarea radacinilor, desenarea graficului functiei, derivarea de ordinul n, integrarea de n ori.

# Bibliografie

* <http://tynerblain.com/blog/2007/04/09/sample-use-case-example/>
* <https://www.code-it.ro/calculator-de-polinoame-in-java/>
* [www.stackoverflow.com](http://www.stackoverflow.com)
* <http://docs.oracle.com/javase/tutorial/collections/algorithms/index.html>
* <http://docs.oracle.com/javase/1.5.0/docs/api/java/util/List.html>