**DOCUMENTATIE**

**TEMA 1**

**NUME STUDENT: Barbu Andrada Corina**

**GRUPA: 30224**

CUPRINS

[1. Obiectivul temei 3](#_Toc130763226)

[1.1. Obiectiv principal: 3](#_Toc130763227)

[1.2. Obective secundare: 3](#_Toc130763228)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc130763229)

[3. Proiectare 4](#_Toc130763230)

[3.1. Structuri de date 4](#_Toc130763231)

[3.2. Diagrama de clase 4](#_Toc130763232)

[3.3. Algoritmi 5](#_Toc130763233)

[4. Implementare 7](#_Toc130763234)

[5. Rezultate 11](#_Toc130763235)

[6. Concluzii 14](#_Toc130763236)

[6.1 Dezvoltari ulterioare 14](#_Toc130763237)

[7. Bibliografie 14](#_Toc130763238)

# Obiectivul temei

## Obiectiv principal:

Principalul obiectiv al acestui proiect este conceperea unui calculator de polinoame, care efectueaza operatii asupra acestora precum: adunare, scadere, inmultire, impartire, derivare si integrare. Interfata grafica a fost conceputa pentru a oferi persoanelor care folosesc aplicatia o utilizare mai usoara a calculatorului, permitandu-i sa adauge doua polinoame sub forma “2x^3+5x^2-3x-2” si sa efectueze operatiile mentionate mai sus asupra lor, rezultatele fiind afisate in casutele corespunzatoare.

## Obective secundare:

Pentru indeplinirea obiectivul principal a fost necesara in primul rand axarea asupra unor obiective secundare, cum ar fi dezvoltarea de use case-uri si scenario, alegerea structurilor de date, impartirea pe clase, dezvoltarea unor algoritmi, implementarea solutiei alese si nu in ultimul rand testarea proiectului.

Obiectivul principal a fost realizat cu:

* Analizarea problemei si dezvoltarea unor scenarii si cazuri
* Alegerea corecta a structurilor de date utilizate
* Impartirea codului in clase (Monom, Polinom)
* Folosirea MVC(Model-View-Controller) pentru o scriere curata si ordonata a codului
* Dezvoltarea unor algoritmi (adaugare monom, parsare string, afisarea polinomului, aflarea celei mai mari puteri, creare polinom, creare lista monoame, validitatea polinomului)
* Implementarea solutiei alese
* Testarea proiectului

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Cel mai important pas inainte de inceperea proiectului a fost stabilirea unor lucruri de baza, adicaa practice a unei fundatii a proiectului. Polinoamele care aveau o singura variabila reprezentau un punct important al proiectului, deoarece de obicei polinomul este o expresie formata din una sau mai multe variabile si constante.

Un alt pas important a fost strangerea si punerea cap la cap a informatiilor acumulate. Deci in cele din urma am ajuns la concluzia ca vom avea nevoie de o clasa Monom si bineinteles, de o clasa Polinom, deoarece cum am spus si mai sus, un polinom este alcatuit din mai multe monoame legate intre ele prin semnul fiecaruia. Fiecare monom are atat un coefficient cat si o putere, deci un hashmap-ul ne este folositor sa stocam toate informatiile de la fiecare monom, iar in cele din urma practice in hashmap sa avem coeficientii si puterile fiecarui monom din polinom. Acest lucru duce la introducerea clasei Operatii, care va efectua operatiile cerute intre polinoame, dar totodata si a celorlalte clase care se bazeaza pe conceptul MVC (prin intermediul caruia facem legatura intre interfata si backend). Totodata nu trebuie sa lipseasca nici prelucrarea datelor, deoarece polinoamele vor fi luate din interfata drept string-uri si din acest motiv am avut nevoie si de o operatie care transforma string-ul in polinom, si in cele din urma in hashmap.

In plus am luat in considerare si faptul ca utilizatorul aplicatiei poate introduce gresit polinoamele, iar in acest caz se trateaza exceptia si se arunca un mesaj de tipul “Polinom invalid”.

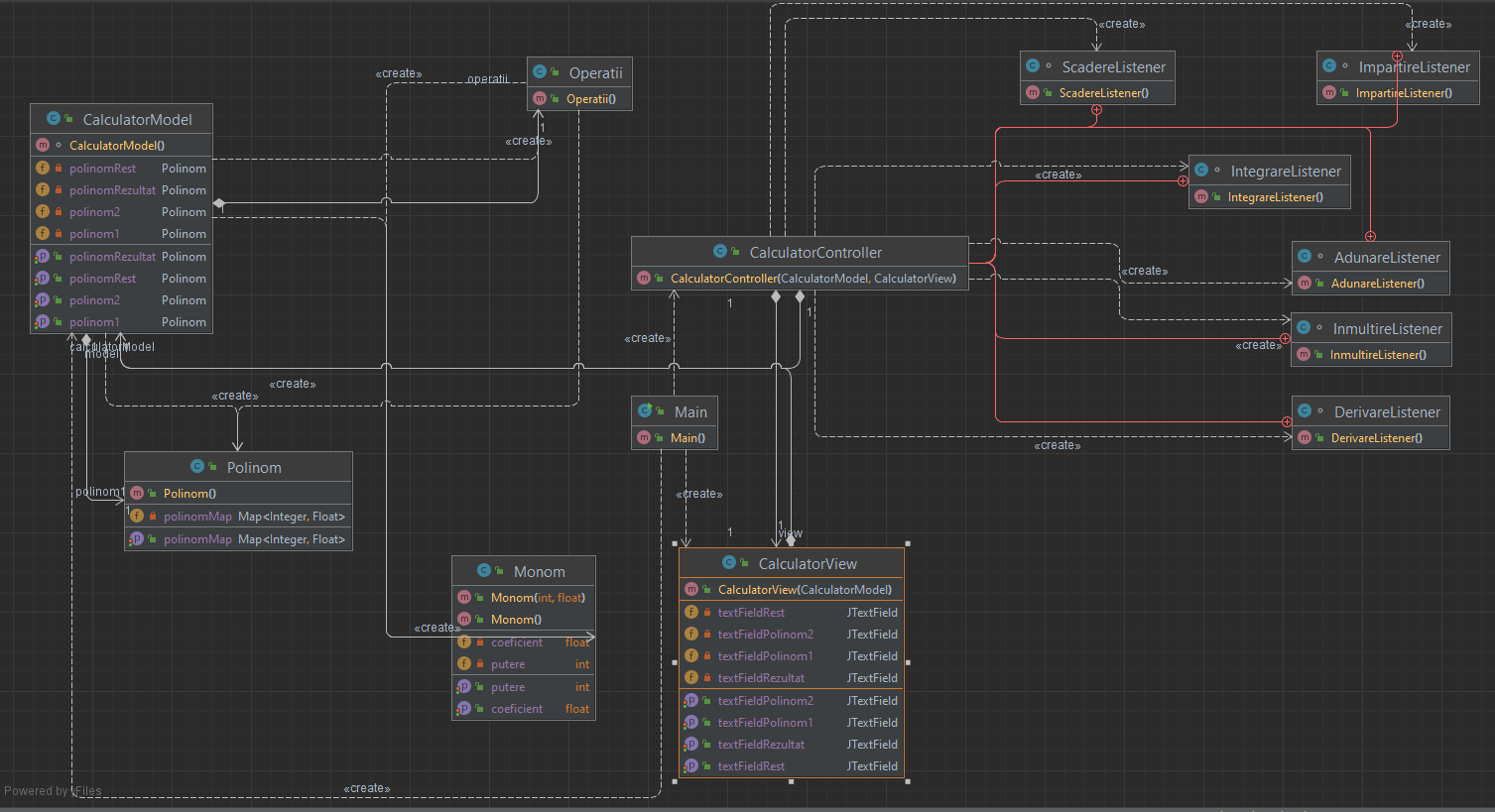
# Proiectare

## Structuri de date

La baza acestui proiect se afla o structura foarte importanta de date, care ne este folositoare pe tot parcursul proiectului si care ne ajuta sa gestionam datele primate de la utilizator mai usor. Hashmap-ul are rolul de a stoca monoamele fiecarui polinom si astfel va stoca in cheie puterea monomului si in valoare, coeficientul monomului. Deci in cele din urma fiecare polinom va avea hashmap-ul sau de monoame in care cheia, adica puterea va fi de tip Integer, iar valoarea, adica coeficientul va fi de tip Float(datorita operatiei de integrare care poate in final sa dea ca si coeficient un numar flotant).

**HashMap<Integer, Float> polinomMap= new HashMap<>();**

## Diagrama de clase



## Algoritmi

* **Algoritmul de adaugare a unui monom: public void addaugareMonom(Monom m)**

Acest algoritm adauga in hashmap cheia si valoarea, respective puterea si coeficientul monomului transmis ca parametru. Se verifica daca daca hashmap-ul nostrum este gol, iar in acest caz, se adauga in el, in caz contrar se verifica daca exista deja cheia pe care dorim sa o adaugam. Daca acea cheie exista, adaugam la valoarea deja existenta a cheii, valoarea(coeficeintul) monomului transmis ca parametru, dar daca nu exista se adauga in hashmap cheia si valoarea monomului.

Se creeaza o lista care o sa contina cheile irelevante, adica cele care au in hash-ul nostru coeficentul 0, dupa care se vor sterge respectivele chei din hash-ul nostru.

* **Algoritmul de afisare a polinomului: String toString()**

Acest algoritm pe langa afisarea polinomului, face si ordonarea acestuia, cu ajutorul unui TreeMap in care vom pune valorile din hashmap-ul nostru.

Iteram noul TreeMap descrescator si iar daca coeficeintul nostru este diferit de 0 luam pe cazuri:

-daca coeficientul este mai mare decat 0, punem automat ‘+’

-daca avem coefficient 1 sau -1, urmat de ‘x’, sa nu-l mai punem

-daca puterea este 1, sa nu o mai afisam

-daca puterea este 0, sa afisam doar coeficientul

-daca caracterul de pe primul index este ‘+’ sa nu-l mai afisam

-daca string-ul este gol, sa se afiseze 0

* **Algoritmul de aflare a puterii cea mai mare a unui monom: public void putereaCeaMaiMare()**

Acest algoritm ne ajuta sa aflam puterea cea mai mare din hashmap-ul nostru de monoame, algoritm care ne va ajuta la impartirea polinaomelor

* **Algoritmul de creare a unui polinom: public Polinom crearePolinom(String stringPolinom)**

Acest algoritm ne ajuta sa cream polinomul, care initial era dat ca parametru drept string.

In aceasta metoda ne vom folosi si de algoritmul spargeInMonoameDeStringuri, de creareListaMonoame si de addaugareMonom, care vor fi descrise mai jos. Se va arunca si o exceptie.

* **Algoritmul de spargere in monoame de stringuri: public List<String> spargeInMonoameDeStringuri(String sir)**

In acest algoritm am folosit un regex pentru a separa fiecare monom din polinom, returnand astfel o lista de string-uri de exemplu:

“2x^2-3x+4” -> “2x^2”, “-3x”, “+4”

In plus se va arunca o exceptie daca monoamele nu sunt corecte.

* **Algoritmul de creare a unei liste de monoame: public List<Monom> creareListaMonoame (List<String> listaSubsiruri)**

Acest algoritm creaza o lista de monoame si sparge fiecare string primit ca parametru in funztie de mai multe cazuri:

-daca nu exista coeficient, sa puna in monom coeficientul 1, iar daca inainte de caracterul “x” s-a gasit doar caracterul “-”, sa puna -1

-daca exista coeficient sa l puna in monom

-daca dupa caracterul “x” urmeaza semn, puterea va fi 1, iar daca x nu exista, puterea sa fie 0

-daca exista si putere dupa “x^”, puterea va avea valoarea respective de dupa caracterul “^”

La sfarsit se pune puterea si coeficientul in monom, dupa care se repeat pasii de mai sus, iar in final se returneaza lista de monoame

* + **Algoritmul de verificare a validitatii polinomului: public Boolean estePolinomulValid(String sir)**

Acesta verifica daca introducerea polinoamelor de catre utilizator s-a efectuat correct, in caz contrar cu ajutorul acestui algoritm se va arunca o exceptie.

# Implementare

Proiectul contine in cele din urma 8 clase: Monom, Polinom, Operatii, CalculatorView, CalculatorController, CalculatorModel, CalculatorTest si bineinteles, clasa Main.

* Clasa **Monom**

Aceasta clasa este una din clasele de baza ale proiectului, fiind astfel caracterizata de doua variabile instanta: putere si coeficient. De nelipsit din implementarea acestei clase este constructorul cu cei doi parametri, metodele de set si get pentru cele doua variabile instanta si metoda compareTo pentru compararea, in functie de putere, a monoamelor.

* Clasa **Polinom**

Aceasta clasa contine un map de monoame, mai exact un hashmap, care practic alcatuieste polinomul, constructorul, metodele de set si get dar si metodele urmatoare:

* public void reinitializarePolinom()
* public void putereaCeaMaiMare()
* public void addaugareMonom(Monom m)
* public String toString()
* public Boolean equals(Object o)
* Clasa **Operatii**

In clasa aceasta am realizat operatiile pe care utilizatorul doreste sa le efectueze asupra polinoamelor. Fiecare operatie in parte se descrie in felul urmator:

* **Adunare**: public Polinom adunarePolinom(Polinom sursaPolinom1, Polinom sursaPolinom2)

-se va crea un nou polinom “polinomRezultat”

-se va itera hashmap-ul primului polinom si se va crea un nou monom care contine cheia si valoarea fiecarui element din hash, dupa care se vor adauga monomamele la polinomRezultat cu ajutorul metodei din polinom addaugareMonom(), apoi la fel si pentru cel de-al doilea polinom

* **Scadere**: public Polinom scaderePolinom(Polinom sursaPolinom1, Polinom sursaPolinom2)

-se va crea un nou polinom “polinomRezultat”

-se va aduna primul polinom la polinomul rezultat, cu ajutorul metodei descries mai sus

-se va itera hashmap-ul celui de-al doilea polinom si se va crea un nou monom care contine cheia si valoarea fiecarui element din hash, dar de aceasta data, valoarea va fi negativa, dupa care se va adauga monoamele la polinomRezultat cu ajutorul metodei din polinom addaugareMonom()

* **Inmultire**: public Polinom inmultirePolinom(Polinom sursaPolinom1, Polinom sursaPolinom2)

- se va crea un nou polinom “polinomRezultat”

-se vor itera hashmap-urile celor doua polinoame, se creeaza un nou monom, care va avea ca putere cheia primului element din hashmap-ul primului polinom adunat cu cheia primului element din hashmap-ul celui de-al doilea polinom

-tot cu ajutorul functiei addaugareMonom() vom aduna monomalele de mai sus la polinomuRezultat

* **Derivare**: public Polinom derivarePolinom(Polinom sursaPolinom)

- se va crea un nou polinom “polinomRezultat”

- se va itera hashmap-ul primului polinom si se va crea un nou monom care contine cheia si valoarea fiecarui element din hash, dupa care se vor adauga monomamele la polinomRezultat cu ajutorul metodei din polinom addaugareMonom(), dar puterea va avea valoarea cheii-1, iar coeficientul va deveni cheia din hashmap inmultita cu valoarea, conform formulei de derivare din matematica.

-inainte de adunarea monoamelor la polinomRezultat, vom pune o conditie de verificare care va adauga monomul doar daca are puterea diferita de -1

* **Integrare**: public Polinom integrarePolinom(Polinom sursaPolinom)

- se va crea un nou polinom “polinomRezultat”

- se va itera hashmap-ul primului polinom si se va crea un nou monom care contine cheia si valoarea fiecarui element din hash, dupa care se vor adauga monomamele la polinomRezultat cu ajutorul metodei din polinom addaugareMonom(), dar puterea de data aceasta va fi cheia incrementa cu unu, iar coeficientul va deveni valoarea impartita la cheia incrementa cu unu

* **Impartire**: public void impartirePolinom(Polinom sursaPolinom1, Polinom sursaPolinom2, Polinom polinomRezultat, PolinompolinomRest)

- principiul acestui algoritm respecta regula din matematica: deimpartit = cat \* impartitor +rest

-pornim de la premisa ca nu se pot imparti, deci restul va fi polinomul1

-luam monoamele cu grad cel mai mare din cele doua polinoame si le comparam

- daca monomul din primul polinom are gradul mai mic decat monomul din cel de-al doilea, atunci nu se poate imparti, deci rezultatul o sa fie 0 iar restul o sa fie polinomul 1; daca nu impartim monoamele iar rezultatul il punem in polinomul rezultat

-vrem sa verificam daca a mai ramas ceva in urma impartirii, deci inmultim rezImpartire cu polinom2 si obtinem restul scazand din restul actual(cel egal cu polinom1) rezultatul inmultirii

-in cazul in care restul actualizat este 0, inseamna ca impartirea s-a facut exact si nu exista rest, altfel se ia monomul cu cea mai mare putere de pe restul actualizat si se compara cu monomul2(cel luat de la polinom2)

-la final se seteaza pe polinomRezultat polinomul rezultat si pe polinomRest restul

* Clasa **CalculatorView**

In aceasta clasa am implementat interfata, cu ajutorul unui form + drag and drop.

Clasa contine elemente care implementeaza interfata grafica, avand astfel:

-4 TextField-uri(doua pentru introducerea polinoamelor si doua pentru afisarea rezultatului si restului necesar la impartire), deci implicit si 4 JLabel-uri

-6 butoane pentru cele 6 operatii

-un JPanel

-ascultatori pentru fiecare buton

-metode set si get

-o metoda (public void aruncaEroare(String s)) care arunca un mesaj de eroare in cazul in care polinoamele nu sunt valide

* Clasa **CalculatorController**

Conform concepului MVC, aceasta clasa face legatura intre CalculatorView, ceea ce vede utilizatorul si CalculatorModel, partea de backend. Tot aici sunt reprezentate actiunile care sunt efectuate atunci cand anumite butoane sunt apasate, alaturi de anumite cazuri de exceptii in cazul in care polinoamele nu au fost introduse correct.

* Clasa **CalculatorModel**

In aceasta clasa este construit backend-ul proiectului. Pentru fiecare operatie am facut cate o metoda in care se initializeaza polinomul rezultat astfel incat operatiile sa nu fie repetate, dupa care in polinomul rezultat se va face operatia dorita din clasa Operatii.

Totodata am implementat si metode de ste si get pentru polinaome, iar pe langa acestea am implementat si metodele urmatoare care au fost descrise la punctul 3.3 Algoritmi:

-public Polinom crearePolinom(String stringPolinom)

-public List<String> spargeInMonoameDeStringuri(String sir)

-public List<Monom> creareListaMonoame(List<String> listaSubsiruri)

-public Boolean estePolinomulValid(String sir)

* Clasa **CalculatorTest**

In aceasta clasa se va testa fiecare operatie pe un exemplu concret conceput de noi la finalul proiectului.

Fiecare operatie va avea atat un test bun, cat si un test rau, pentru a testa ma bine functionalitatea programului.

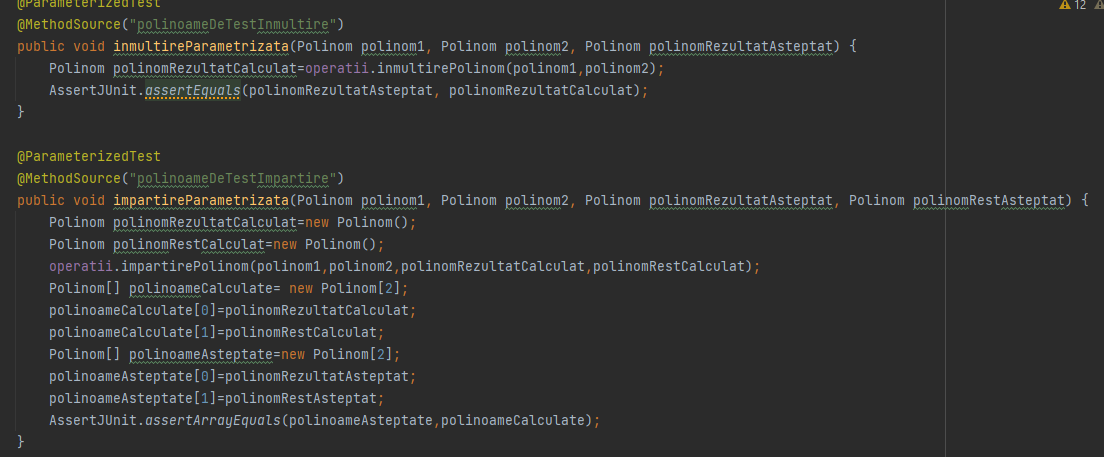
* Clasa **Main**

Reprezinta clasa care face legatura intre celelalte clase.

# Rezultate

Testele sunt facute in clasa CalculatorTest, in care am facut metode pentru fiecare operatie, testand astfel rezultatul bun, dat de noi, cu rezultatul in urmal operatiei efectuate din clasa Operatii. Totodata pentru fiecare operatie am facut un test bun si unul rau. Testele sunt parametrizate la fel cum ni s-a dat in Assigment3. Am suprascris metoda equals in clasa Polinom pentru a putea compara chiar obiectul rezultatului asteptat cu cel calculate si nu string-ul. Vom exemplifica procedeul de testare pentru cateva operatii in figurile de mai jos.





O imagine care conține text

Descriere generată automat

O imagine care conține text

Descriere generată automat

# Concluzii

In cele din urma, datorita implementarii acesturi proiect consider ca mi-am imbunatatit abilitatile de scriere a codului in JAVA si totodata abilitatea de a imparti codul in mai multe clase, astfel incat sa fie cat mai lizibil. In plus, fapul ca am retinut datele polinomului intr-un hashmap, m-a ajutat sa gestionez memoria mai bine.

## 6.1 Dezvoltari ulterioare

* Imbunatarirea, infrumusetarea si dezvoltarea interfetei grafice
* Calculul derivatei de ordin n, unde n este un numar natural introdus de utilizator

# Bibliografie

[https://www.w3schools.com/java/java\_regex.asp#](https://www.w3schools.com/java/java_regex.asp)

<http://stackoverflow.com/questions>

<https://dev.java/learn/pattern-matching/>

<https://www.geeksforgeeks.org/sorting-hashmap-according-key-value-java/>

<https://msdn.microsoft.com/en-us/library/>