# DOCUMENTATIE

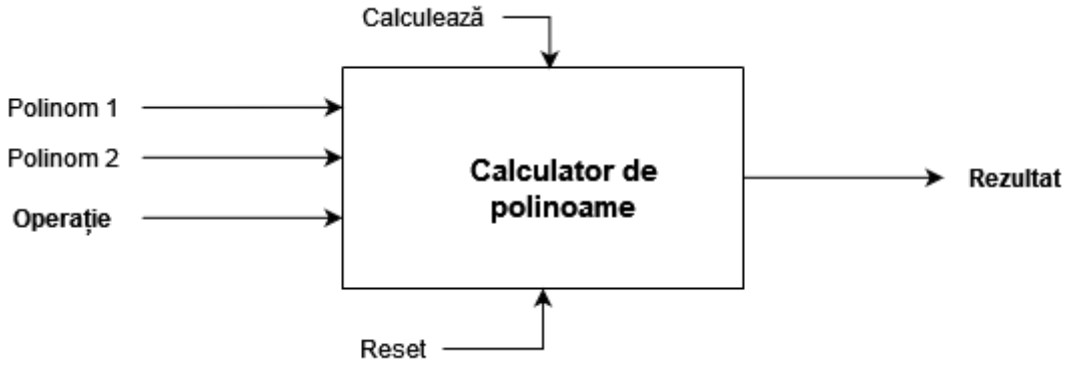
## TEMA *1*

NUME STUDENT: Bartolomei Vlad-Alexandru
GRUPA: 30229

# CUPRINS

# 1. Obiectivul temei

Obiectivul principal al prezentului calculator de polinoame este de a automatiza operațiile aferente a două polinoame, care pe hârtie sunt mai greu de făcut.

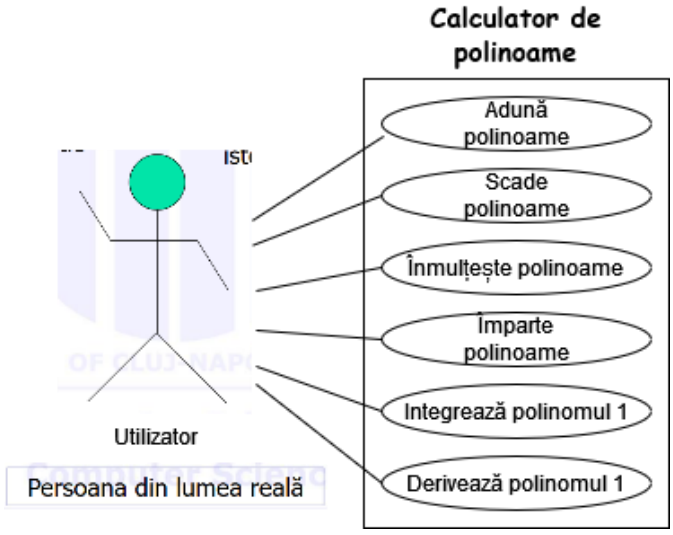



Obiective secundare:

- Analiza problemei, public țintă, modul în care un asemenea dispozitiv (descris ca un black box, mai sus) ar putea fi utilizat -> Capitolul 2
- Proiectarea interfeței grafice și a implementării funcționalităților. Cerințele (non-) funcționale se găsesc în Capitolul 2, iar împărțirea pe pachete și clase, precum și modul în care interacționează clasele, se găsesc în Capitolul 3.
- Descrierea într-un limbaj natural a celor mai importante metode și a implementării interfeței grafice, pe scurt. Se găsește în Capitolul 4.
- Observarea rezultatelor (Capitolul 5) și tragerea concluziilor în urma implementării, însoțite de posibile viitoare îmbunătățiri ale aplicației (Capitolul 6)
- Punerea în evidență a celor mai importante resurse bibliografice (Capitolul 7).

# 2. Analiza problemei, modelare, scenariu, cazuri de utilizare

## *2.1. Diagramă Use-Case*



**Use case:** [orice operație, în afară de integrare sau derivare] polinoamele

**Actor principal:** persoana din viața reală

**Scenariul principal de succes:**

**A)** Persoana introduce două polinoame în interfața grafică furnizată.

**B)** Utilizatorul selectează operația dorită: adunare, scădere, înmulțire, împărțire.

**C)** Utilizatorul apasă butonul „Calculează”

**D)** Sistemul efectuează operația, iar rezultatul este afișat într-o zonă de „Rezultat” a Interfeței Grafice.

**Scenarii Alternative:** Polinoame Incorecte

- Persoana nu completează cele două câmpuri de polinoame cu polinoame corecte. Nu se acceptă ca utilizatorul să introducă un singur polinom. => ne întoarcem la pasul A)

**Use case:** [Integrează sau derivă] polinoamele

**Actor principal:** persoana din viața reală

**Scenariul principal de succes:**

**A)** Persoana introduce un polinom în interfața grafică furnizată, în zona dedicate primului polinom.

**B)** Utilizatorul selectează operația dorită: Integrare, derivare.

**C)** Utilizatorul apasă butonul „Calculează!”

**D)** Sistemul efectuează operația, iar rezultatul este afișat într-o zonă de „Rezultat” a Interfeței Grafice.

**Scenarii Alternative:** Polinom introdus în locul greșit

- Persoana introduce polinomul dorit spre derivare în spațiul destinate celui de al doilea polinom =: ne întoarcem la pasul A)

Polinom incorrect introdus

- Persoana completează primul câmp de polinoame cu un polinom incorect. => ne întoarcem la pasul A)

În continuare, vom referi la Calculatorul de Polinoame prin termenul de „sistem”

**Cerințe funcționale:**

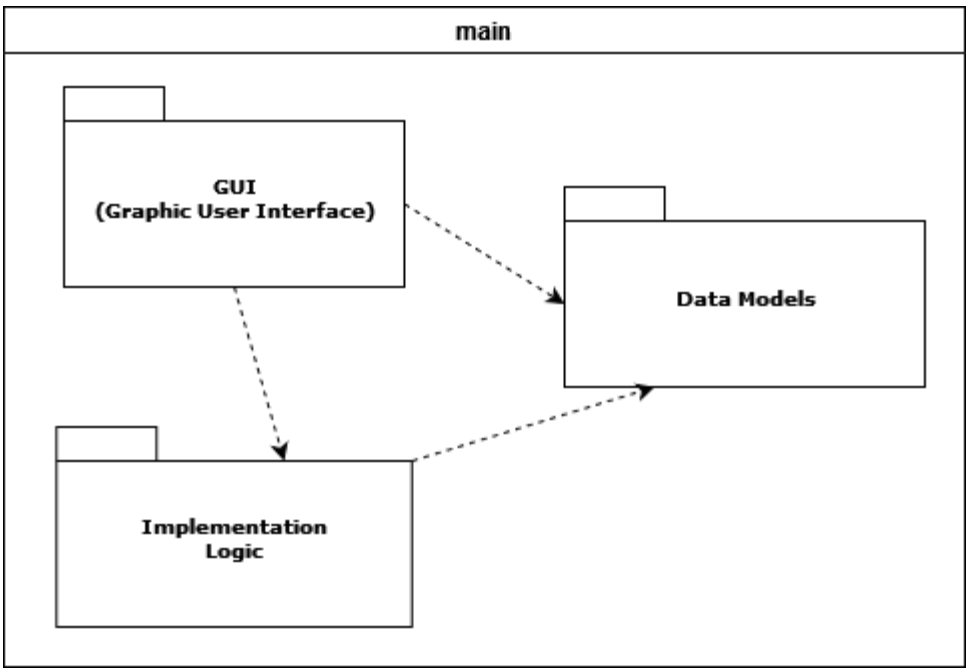
- Sistemul ar trebui să permită utilizatorului să introducă două polinoame sau, după caz, unul singur
- Sistemul ar trebui să permită utilizatorului să selecteze operația matematică
- Sistemul ar trebui să permită utilizatorului să efectueze operația selectată
- Sistemul ar trebui să permită utilizatorului să reseteze câmpurile editabile și selectabile prin apăsarea butonului „Resetare”
- Sistemul ar trebui să permită utilizatorului să termine execuția programului prin apăsarea butonului de închidere al ferestrei
- Sistemul nu ar trebui să permită utilizatorului extinderea ferestrei

**Cerințe non-funcționale:**

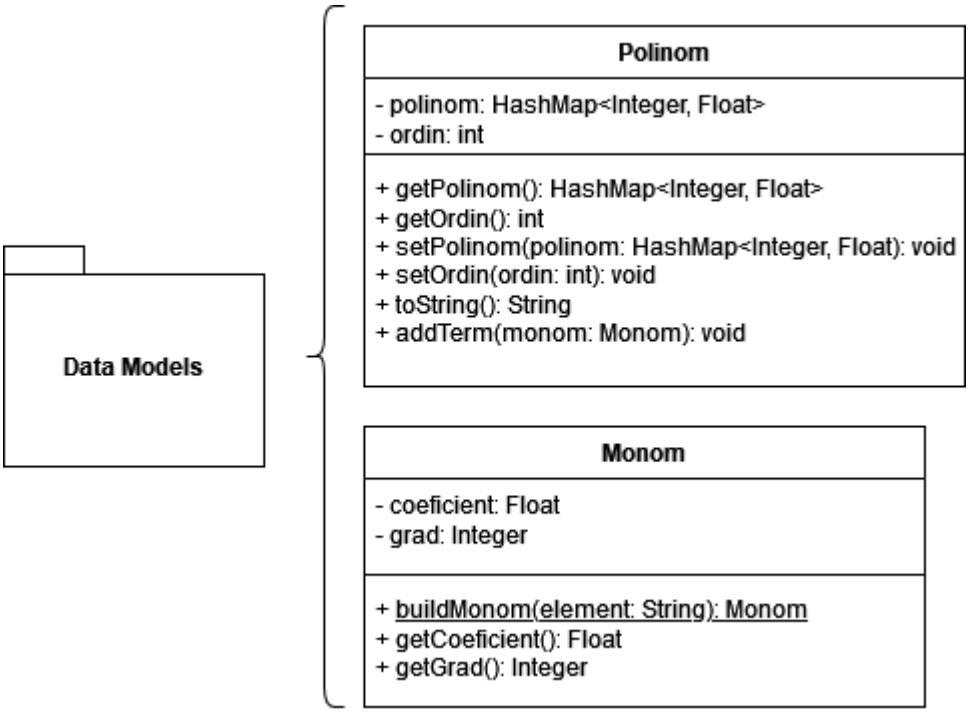
- Sistemul ar trebui să fie intuitiv și ușor de utilizat de utilizator
- Sistemul ar trebui să aibă un aspect atractiv pentru utilizator

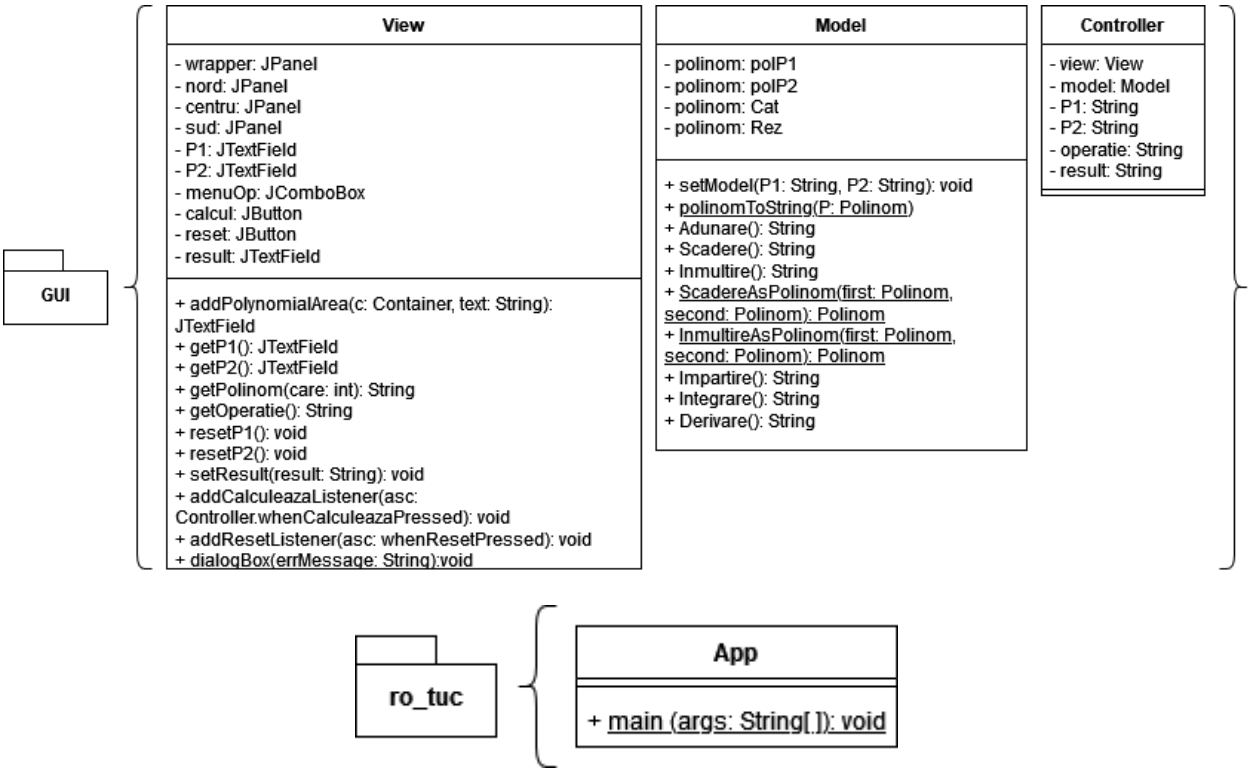
# 3. Proiectare

## *3.1. Diagramă de pachete*



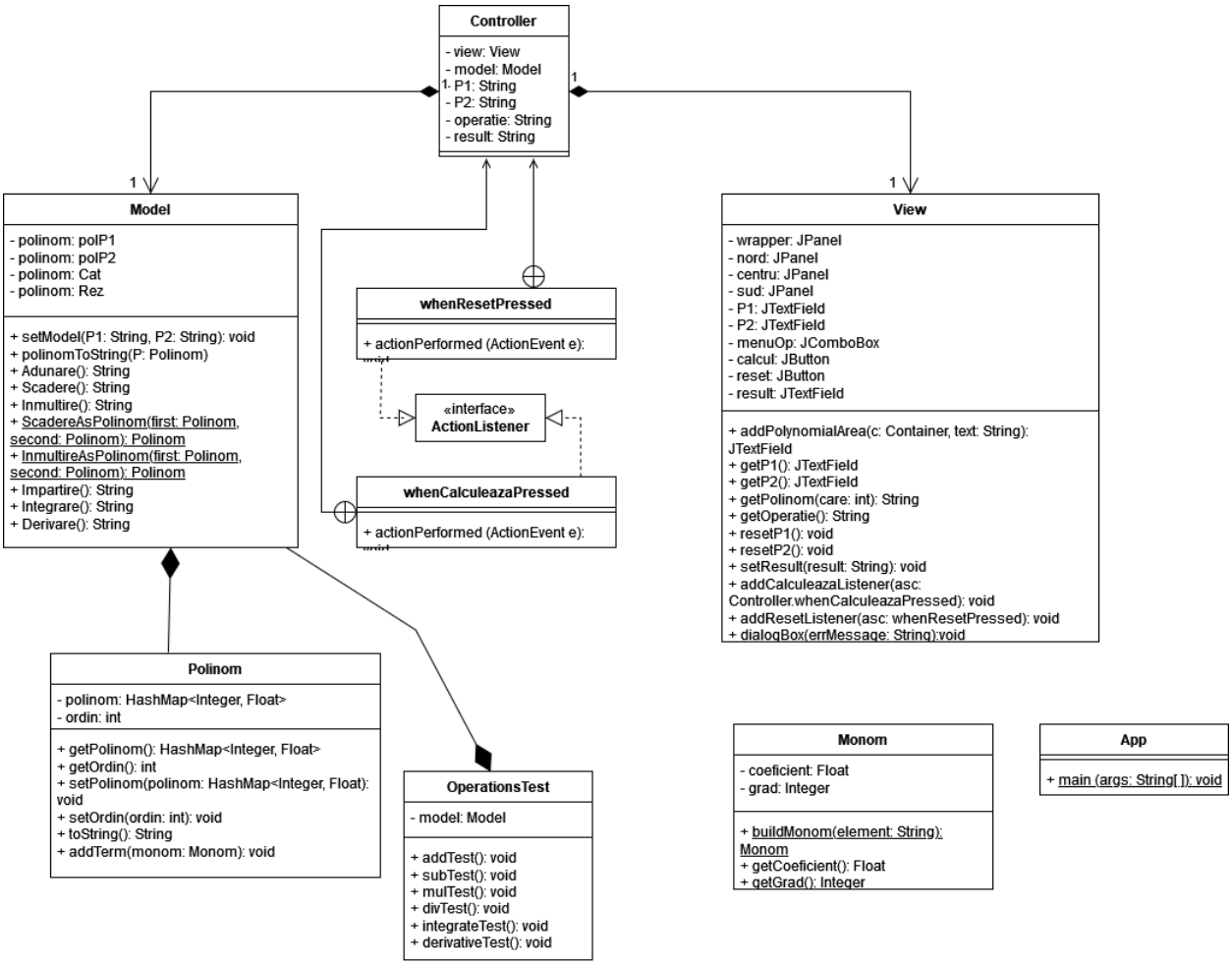

## *3.2. Conținutul fiecărui pachet*




GUI
View
- wrapper: JPanel
- nord: JPanel
- centru: JPanel
- sud: JPanel
- P1: JTextField
- P2: JTextField
- menuOp: JComboBox
- calcul: JButton
- reset: JButton
- result: JTextField
+ addPolynomialArea(c: Container, text: String): JTextField
+ getP1(): JTextField
+ getP2(): JTextField
+ getPolinom(care: int): String
+ getOperatie(): String
+ resetP1(): void
+ resetP2(): void
+ setResult(result: String): void
+ addCalculeazaListener(asc: Controller.whenCalculeazaPressed): void
+ addResetListener(asc: whenResetPressed): void
+ dialogBox(errMessage: String):void
Model
- polinom: polP1
- polinom: polP2
- polinom: Cat
- polinom: Rez
+ setModel(P1: String, P2: String): void
+ polinomToString(P: Polinom)
+ Adunare(): String
+ Scadere(): String
+ Inmultire(): String
+ ScadereAsPolinom(first: Polinom, second: Polinom): Polinom
+ InmultireAsPolinom(first: Polinom, second: Polinom): Polinom
+ Impartire(): String
+ Integrare(): String
+ Derivare(): String
Controller
- view: View
- model: Model
- P1: String
- P2: String
- operatie: String
- result: String


ro_tuc
App
+ main (args: String[ ]): void

### *3.3. Diagrama de clase*




# 4. Implementare

Pe parcursul întregii implementări a proiectului, două funcții au jucat un rol major.

### 4.1. Funcția addTerm

Specifică clasei Polinom.

```
13 usages   new *
public void addTerm(Monom monom) {
    Integer grad = monom.getGrad();
    Float coef = monom.getCoeficient();
    int ok = -1;
    for (Integer i : polinom.keySet()) {
        if (i == grad)
            ok = i; // in ok retinem gradul, daca acesta coincide cu cel al monomului dat
        // sau -1 daca monomul dat are un grad care nu exista
        if (i > ordin)
            ordin = i;
    }
    if (ok == -1) {
        polinom.put(grad, coef);
        if (grad > ordin)
            ordin = grad;
    } else {
        Float var = polinom.get(ok);
        var += coef;
        polinom.remove(ok);
        polinom.put(ok, var);
    }
}
```

Această metodă primește ca parametru un monom de clasă monom, rezultat în urma obținerii termenilor cu ajutorul expresiilor regulate, și adaugă la polinomul *polinom* din clasa curentă (este un atribut) monomul. Dacă gradul acestui monom transmis există deja printre gradele polinomului curent, se modifică doar coeficientul; altfel se adaugă monomul cu totul.

Metoda mai are o funcționalitate: pentru polinomul nou format, determină în timp real ordinul maxim al acestuia.

### 4.2. Funcția Împărțire

Specifică clasei Model.

```
2 usages   new *
public String Impartire(){
    Rez.getPolinom().clear(); Rez.setOrdin(0); // vom folosi Rez pe post de polinom pentru Rest
    Cat.getPolinom().clear(); Cat.setOrdin(0);
    int ordMax1 = polP1.getOrdin();
    int ordMax2 = polP2.getOrdin();
    if(ordMax2 == 0)
        if(ordMax1 == 0)
            return "Eroare 2";
        else return "Eroare";
    while( ordMax2 <= ordMax1 ){
        Monom monom = new Monom( coeficient: polP1.getPolinom().get(ordMax1) / polP2.getPolinom().get(ordMax2),
                grad: ordMax1 - ordMax2);
        Cat.addTerm(monom);
        Polinom aux = new Polinom();
        aux.getPolinom().clear();
        aux.addTerm(monom);
        Rez = InmultireAsPol(polP2, aux);
        polP1.setPolinom(ScadereAsPol(polP1, Rez).getPolinom());

        ordMax1--;
    }
    return "Cât: "+ polinomToString(Cat) + " || Rest: " + polinomToString(polP1);
}
```

Metoda respectă algoritmul de împărțire a două polinoame. Inițial aduce în ordMax1 și ordMax2 ordinele celor două polinoame introduse de utilizator. Mai apoi testează cazurile problemă: dacă cumva avem 0/0 sau Polinom/0, i.e. împărțiri ilegale. În caz contrar, controlul continua cu implementarea, care se folosește de două polinoame noi, Cat (Cât) și Rez (Rest).

## Despre implementarea interfeței utilizator:

Pentru aceasta s-a optat pentru folosirea unui pattern de tip Model-View-Controller.

Cum ar fi de așteptat, View conține o fereastră cu un panou principal wrapper, care are ca principală caracteristică un BorderLayout. Din moment ce au fost necesare doar regiunile Nord, Centru și Sud, trei containere (panouri) cu aceeași denumire și-au făcut loc printre liniile de cod. Pentru selectarea operațiilor, am ales un JComboBox, selectorul purtând denumirea de menuOp. Interfața grafică mai dispune de două butoane a căror funcționalități sunt intuitive: Calculează și Reset.

Pentru a nu strica aspectul vizual, aplicația nu este redimensionabilă.

În clasa Model vom regăsi atât implementarea operațiilor polinomiale cerute de utilizator, cât și metoda polinomToString, care va transforma HashMap-ul polinom într-un String necesar afisării în JTextField-ul Result din View.

Clasa Control controlează atât clasa View, cât și clasa Model. *Control* impune clasei View doi ascultători, ai butoanelor Calculează și Reset. În ascultătorul butonului Calculează, metoda actionPerformed testează natura *operației* recepționate în constructorul lui Model, venind din View (cu un getter), și pe baza acesteia, cheamă din Model metoda aferentă operației dorite, urmând a redirecta rezultatul (care vine direct sub formă de string, grație metodei polinomToString) către Interfața Grafică (în View).

# 5. Rezultate

În cadrul proiectului au fost făcute o serie de teste relativ la fiecare dintre cele 6 operații de care sistemul este capabil. În materie de coeficienți, precizia acestora este specifică obiectelor de tipul clasei învelitore Float, astfel că nu ar trebui să ne mire că un coeficient precum 3 (din input) se transformă în 3.0 pe output. Deoarece calculele matematice corespund cu rezultatele așteptate, sistemul a trecut toate testele și este gata de utilizare.

# 6. Concluzii

În urma realizării acestei teme, au fost atinse concepte matematice legate de polinoame și operațiile de bază cu acestea. Un adevărată provocare a constituit-o traducerea unui polinom din text introdus într-un obiect modelat în forma dorită (clasa polinom), precum și implementarea operațiilor, cu precădere cea de împărțire, întrucât algoritmul nu e chiar trivial matematic vorbind. Noțiuni despre implementarea graficp a unei aplicații au fost reamintite cu succes.

Există suficient loc de îmbunătățiri. Aplicația se poate îmbunătăți în primul rând grafic, autorul proiectului propunând o afișare în format matematic a polinoamelor introduse și a celui rezultat. De asemenea, în partea de jos a ferestrei, o zonă de mesaje ar putea fi implementată, de tip read-only, care să anunțe utilizatorul în momentul în care input-ul în sistemul de față nu corespunde / nu poate fi tradus pe înțelesul codului aplicației.

# 7. Bibliografie


1. *Interfețe utilizator grafice (GUIs), Curs POO, conf. prof. dr. Ion Augustin Giosan - https://users.utcluj.ro/~igiosan/Resources/POO/Curs/POO09.pdf*
2. *Clase învelitoare. Operatori. Structuri de control în Java. Clase și obiecte, Curs POO, conf. prof. dr. Ion Augustin Giosan - https://users.utcluj.ro/~igiosan/Resources/POO/Curs/POO02.pdf*
3. *Create gradient translucent windows in Java Swing - https://www.tutorialspoint.com/Create-gradient-translucent-windows-in-Java-Swing*
4. *How to iterate HashMap in Java - https://www.geeksforgeeks.org/how-to-iterate-hashmap-in-java/*
5. *Regex for polynomial expressions - https://stackoverflow.com/questions/36490757/regex-for-polynomial-expression*