DOCUMENTAȚIE

TEMA 1

Nume student: VELICEA ANDREEA – IOANA

Grupa: 30228

Contents

1. Obiectivul temei

Principalul obiectiv al acestei teme este designul si implementarea unui calculator de polinoame, acestea sa fiind de o singura variabila cu coeficienti reali. Acesta ar trebuie sa contina o interfata grafica “User friendly”, prin care utilizatorul poate insera polinoame, poate selecta o operatie matematica si poate afisa rezultatul.

1.1 Obiective secundare

- Analizarea problemei si identificarea cerintelor – Capitolul 2
- Proiectarea calculatorului de polinoame – Capitolul 3
- Implementarea calculatorului de polinoame – Capitolul 4
- Testarea calculatorului de polinoame – Capitolul 5

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

2.1 Cerinte functionale

- Calculatorul de polinoame ar trebui sa permita utilizatorului sa introduca polinoame.
- Calculatorul de polinoame ar trebui sa permita utilizatorului sa selecteze operatie matematica pe care acesta doreste.
- Calculatorul de polinoame ar trebui sa adauge doua polinoame.
- Calculatorul de polinoame ar trebui sa afiseze rezultatul operatiei alese.
- Calculatorul de polinoame ar trebui sa permita utilizatorului sa reintroduca alte polinoame.

2.2 Cerinte non functionale

- Calculatorul de polinoame ar trebui sa fie intuitiv si usor de utilizat de catre utilizator
- Calculatorul de polinoame ar trebui sa aiba un aspect placut.

2.3 Prezentarea use-case-urilor

Use case - aduna polinoame

Actor principal : utilizatorul

Scenariu principal in caz de succes :

- Utilizatorul insereaza fiecare polinom pe rand si apasa butonul " Insereaza Polinom" pentru a putea si procesat fiecare dintre cele doua polinoame
- Utilizatorul apasa butonul « ADUNARE »
- Calculatorul de polinoame proceseaza cele doua polinoame inserate, calculeaza suma lor si afiseaza rezultatul.

Scenariu in caz de esec : Polinoame incorecte

- Utilizatorul introduce 2 polinoame incorecte (de exemplu daca acesta introduce un polinom de mai multe variabile)
- Calculatorul afiseaza un pop-up de eroare si utilizatorul trebuie sa introduca alte polinoame.

Use case - scade polinoame

Actor principal : utilizatorul

Scenariu principal in caz de succes :

- Utilizatorul insereaza fiecare polinom pe rand si apasa butonul " Insereaza Polinom" pentru a putea si procesat fiecare dintre cele doua polinoame
- Utilizatorul apasa butonul « SCADERE »
- Calculatorul de polinoame proceseaza cele doua polinoame inserate, calculeaza scaderea lor si afiseaza rezultatul.

Scenariu in caz de esec : Polinoame incorecte

- Utilizatorul introduce 2 polinoame incorecte (de exemplu daca acesta introduce un polinom de mai multe variabile)

- Calculatorul afiseaza un pop-up de eroare si utilizatorul trebuie sa introduca alte polinoame.

Use case - inmulteste polinoame

Actor principal : utilizatorul

Scenariu principal in caz de succes :

- Utilizatorul insereaza fiecare polinom pe rand si apasa butonul " Insereaza Polinom" pentru a putea si procesat fiecare dintre cele doua polinoame
- Utilizatorul apasa butonul « INMULTESTE »
- Calculatorul de polinoame proceseaza cele doua polinoame inserate, calculeaza produsul lor si afiseaza rezultatul.

Scenariu in caz de esec : Polinoame incorecte

- Utilizatorul introduce 2 polinoame incorecte (de exemplu daca acesta introduce un polinom de mai multe variabile)
- Calculatorul afiseaza un pop-up de eroare si utilizatorul trebuie sa introduca alte polinoame.

Use case - imparte polinoame

Actor principal : utilizatorul

Scenariu principal in caz de succes :

- Utilizatorul insereaza fiecare polinom pe rand si apasa butonul " Insereaza Polinom" pentru a putea si procesat fiecare dintre cele doua polinoame
- Utilizatorul apasa butonul « IMPARTE »
- Calculatorul de polinoame proceseaza cele doua polinoame inserate, calculeaza impartirea lor si afiseaza rezultatul repartizat in catul si restul impartirii

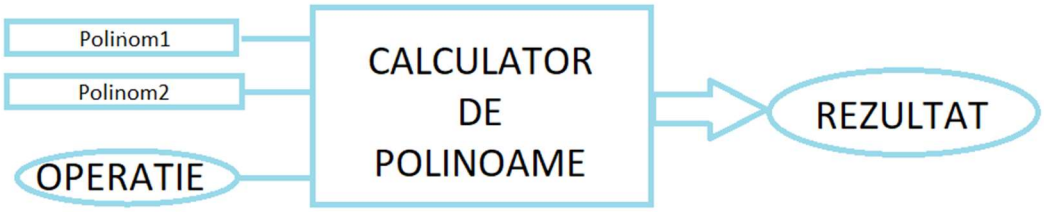
Scenariu in caz de esec : Polinoame incorecte

- Utilizatorul introduce 2 polinoame incorecte (de exemplu daca acesta introduce un polinom de mai multe variabile)
- Calculatorul afiseaza un pop-up de eroare si utilizatorul trebuie sa introduca alte polinoame.

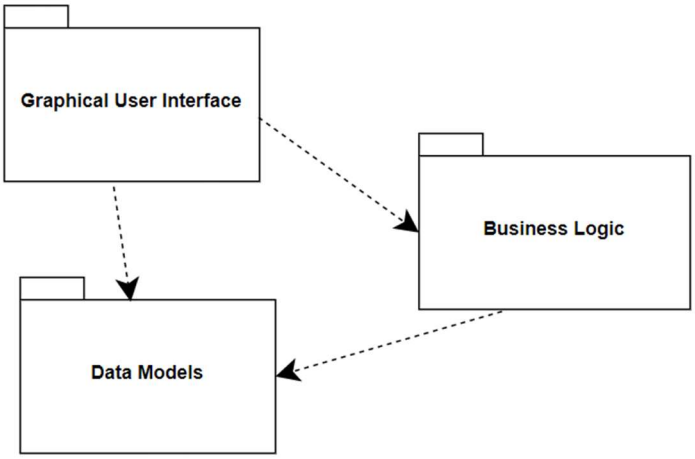
Scenariu in caz de esec : Gradul polinomului 2 mai mic decat gradul polinomului 1

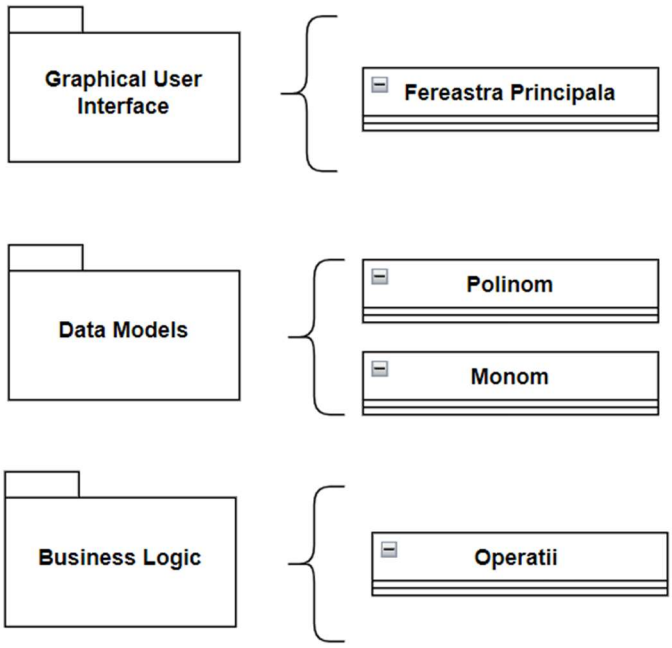
- Utilizatorul introduce 2 polinoame incorecte din punctul de vedere al inmultirii. Acesta introduce un polinom de grad mai mare ca si primul polinom, si un polinom cu grad mai mic ca si al doilea polinom.
- Calculatorul afiseaza un pop-up de eroare si utilizatorul trebuie sa introduca alte polinoame.

3. Proiectare

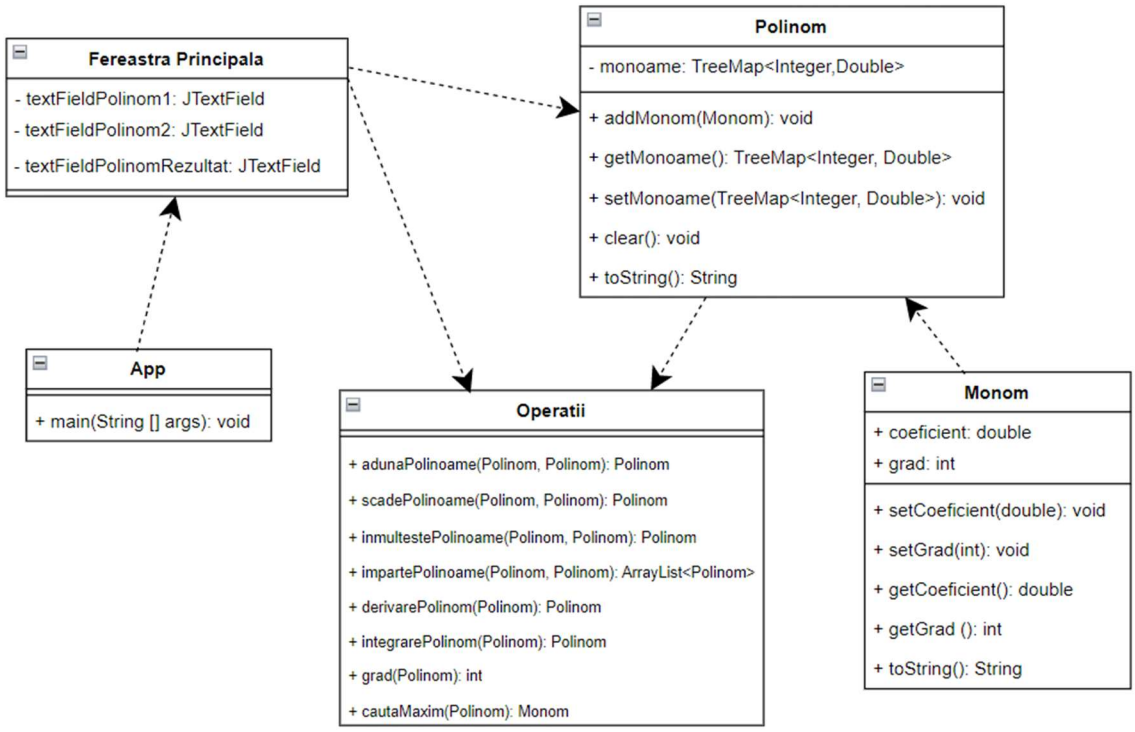


3.2 Diagrama de PACHETE





3.3 Diagrama de CLASE



3.4 Algoritmi folositi

- Pentru realizarea operatiei de impartire am folosit un algoritm specific impartirii polinoamelor

```
function n / d is
    require d ≠ 0
    q ← 0
    r ← n             // At each step n = d × q + r

    while r ≠ 0 and degree(r) ≥ degree(d) do
        t ← lead(r) / lead(d)       // Divide the leading terms
        q ← q + t
        r ← r − t × d

    return (q, r)
```

3.4 Structuri de Date folosite

- pentru maparea Monoamelor in polinoame, am folosit Colectia TreeMap pentru a putea sorta in ordinea key-lor (ordinea gradelor) fiecare polinom in parte. De asemenea am folosit un Compator pentru a le mapa in ordinea descrescatoare a gradelor si a afisa polinomul de la gradul cel mai mare la gradul cel mai mic.

4. Implementare

Clasa Monom

- Este clasa in care este definit un monom cu gradul si coeficientul lui.
- Are ca si atribute :
    - Coeficientul, care este de tip Double
    - Gradu, care este de tip Integer
- Constructorul clasei are rolul de a instantia obiecte pentru clasa Monom si are antetul :
    - Public Monom(double coeficient, int grad) ;
- Metode importante pe care clasa Monom le contine sunt :

```
public void setCoeficient(double coeficient) {
    this.coeficient = coeficient;
}
```

    - aceasta metoda ne permite sa modificam valoarea coeficientului

```
public void setGrad(int grad) {
    this.grad = grad;
}
```

- aceasta metoda ne permite modificarea valorii gradului

```
public int getGrad() {
    return grad;
}
```

- aceasta metoda ne permite accesarea campului grad

```
public double getCoeficient() {
    return coeficient;
}
```

- aceasta metoda ne permite accesarea campului coeficient

```
public String toString(){
    return coeficient+"*x^"+grad;
}
```

- de asemenea mai avem si metoda suprascris toString() pentru a putea afisa intr un mod mai placut monomul

Clasa Polinom

- in aceasta clasa folosim un TreeMap pentru mapareaza polinoamelor
- are ca si atribute Colectia de monoame :

```
private TreeMap <Integer,Double> monoame = new
TreeMap<>(Collections.<Integer>reverseOrder());
```

- metodele pe care aceasta clasa le continue sunt :

```
public void addMonom(Monom monom){
    if(monoame.containsKey(monom.getGrad())==false) {
        monoame.put(monom.getGrad(), monom.getCoeficient());
    }else{
        monoame.put(monom.getGrad(),
monom.getCoeficient()+monoame.get(monom.getGrad()));
    }
}
```

- aceasta metoda este folosita pentru adaugarea monoamelor in Colectia de monoame.

```
public TreeMap<Integer, Double> getMonoame() {
    return monoame;
}


public void setMonoame(TreeMap<Integer, Double> monoame) {
    this.monoame = monoame;
}
```

- cele doua metode de setare si preluare de monoame prin care putem accesa colectia specifica fiecarui polinom

```
public void clear(){
    monoame.clear();
}
```

- prin metoda clear golim colectia de polinoame
- aceasta metoda este folosita in momentul in care vrem sa inseram noi polinoame si eliberam continutul fiecarui polinom pentru a stoca valorile noi

```
-   public String toString(){..}
```

- aceasta metoda ajuta la afisarea intr un mod mai frumos a polinoamelor in care sunt tratate diferitele cazuri in care gradul sau coeficientul unui monom pot fi 0 sau 1.

Clasa Operatii

- in aceasta clasa sunt prezente operatiile pe polinoame pe care le efectuam, cum ar fi:

o adunarea polinoamelor:

```
-   public Polinom adunaPolinoame(Polinom polinom1, Polinom
    polinom2){
        Polinom rezultat = new Polinom();
        if(polinom2.getMonoame().size()==0 &&
    polinom1.getMonoame().size()==0)rezultat.addMonom(new
    Monom(0,0));
        for(Map.Entry<Integer,Double> entry:
    polinom1.getMonoame().entrySet()){

    if(polinom2.getMonoame().containsKey(entry.getKey())==true){
                Monom monom = new Monom
    (entry.getValue()+polinom2.getMonoame().get(entry.getKey()),en
    try.getKey());
                rezultat.addMonom(monom);
            }
        }
        for(Map.Entry<Integer,Double> entry:
    polinom1.getMonoame().entrySet()){

    if(rezultat.getMonoame().containsKey(entry.getKey())==false){
                rezultat.addMonom(new Monom(entry.getValue(),
    entry.getKey()));
            }
        }
        for(Map.Entry<Integer,Double> entry:
    polinom2.getMonoame().entrySet()){

    if(rezultat.getMonoame().containsKey(entry.getKey())==false){
                    rezultat.addMonom(new Monom(entry.getValue(),
    entry.getKey()));
            }
        }
        return rezultat;
    }
```

- parcurgem fiecare polinom in parte si in momentul in care gasim acelasi grad, adunam coeficientii fiecarui monom in parte si ii adaugam unui alt polinom

rezultat. Mai apoi parcurgem fiecare polinom in parte si verificam elementele care nu au fost comune cu cele din polinomul celalalt si le adaugam rezultatului

- aceasta metoda returneaza polinomul rezultat in urma adunarii

- scadere polinoamelor

```
-   public Polinom scadePolinoame(Polinom polinom1,Polinom
    polinom2){
        Polinom rezultat = new Polinom();
        if(polinom2.getMonoame().size()==0 &&
    polinom1.getMonoame().size()==0)rezultat.addMonom(new
    Monom(0,0));
        for(Map.Entry<Integer,Double> entry:
    polinom1.getMonoame().entrySet()){

    if(polinom2.getMonoame().containsKey(entry.getKey())==true){
                Monom monom = new Monom (entry.getValue()-
    polinom2.getMonoame().get(entry.getKey()),entry.getKey());
                rezultat.addMonom(monom);
            }
        }
        for(Map.Entry<Integer,Double> entry:
    polinom1.getMonoame().entrySet()){

    if(rezultat.getMonoame().containsKey(entry.getKey())==false){
                rezultat.addMonom(new
    Monom(entry.getValue(),entry.getKey()));
            }
        }
        for(Map.Entry<Integer,Double> entry:
    polinom2.getMonoame().entrySet()){

    if(rezultat.getMonoame().containsKey(entry.getKey())==false){
                rezultat.addMonom(new Monom(-
    entry.getValue(),entry.getKey()));
            }
        }
        return rezultat;
    }
```

- parcurgem fiecare polinom in parte si in momentul in care gasim acelasi grad, scadem coeficientii fiecarui monom in parte si ii adaugam unui alt polinom rezultat. Mai apoi parcurgem fiecare polinom in parte si verificam elementele care nu au fost comune cu cele din polinomul celalalt si le adaugam rezultatului
- aceasta metoda returneaza polinomul rezultat in urma scaderii

- inmultirea polinoamelor :

```
-   public Polinom inmultestePolinoame(Polinom polinom1,Polinom
    polinom2){
        Polinom rezultat = new Polinom();
        if(polinom2.getMonoame().size()==0 &&
    polinom1.getMonoame().size()==0)rezultat.addMonom(new
    Monom(0,0));
        for(Map.Entry<Integer,Double> entry1:
    polinom1.getMonoame().entrySet()){
            for(Map.Entry<Integer,Double> entry2:
```

```
    polinom2.getMonoame().entrySet()){
                Monom monom = new
    Monom(entry1.getValue()*entry2.getValue(),entry1.getKey()+entr
    y2.getKey());
                if(monom.getCoeficient()!=0) {

    if(rezultat.getMonoame().containsKey(monom.getGrad())==true){

    monom.setCoeficient(rezultat.getMonoame().get(monom.getGrad())
    +monom.getCoeficient());
                    }
                    rezultat.addMonom(monom);
                }
            }
        }
        return rezultat;
    }
```

- parcurgem fiecare polinom in parte si inmultim fiecare monom din polinomul1, cu fiecare element din polinomul2 si le adaugam la polinomul rezultat.
- Aceasta metoda returneaza polinomul rezultat in urma impartirii celor doua polinoame

- Impartirea polinoamelor:

```
-   public ArrayList<Polinom> impartePolinoame(Polinom polinom1,
    Polinom polinom2)throws Exception{
        ArrayList<Polinom>rezultat = new ArrayList<Polinom>();
        if(polinom2.getMonoame().size()==0 ||
    polinom1.getMonoame().size()==0)throw new Exception("nu se pot
    imparti doua polinoame nule!");
        Polinom cat = new Polinom();
        Polinom rest = polinom1;

        if(grad(rest)< grad(polinom2)){
            throw new Exception("gradul deimpartitului trebuie sa
    fie mai mare decat gradul impartitorului!");
        }else{
            while( grad(rest) >= grad(polinom2)){
                Monom monom1 = cautaMaxim(rest);
                Monom monom2 = cautaMaxim(polinom2);
                Monom monomRezultat = new
    Monom((monom1.getCoeficient())/(monom2.getCoeficient()),
    (monom1.getGrad()-monom2.getGrad()) );
                cat.addMonom(monomRezultat);
                Polinom polinomRezultat = new Polinom();
                polinomRezultat.addMonom(monomRezultat);
                Polinom rezultatInmultire =
    inmultestePolinoame(polinomRezultat,polinom2);
                rest = scadePolinoame(rest, rezultatInmultire);
            }
        }
        rezultat.add(cat);
        rezultat.add(rest);
        return rezultat;
    }
```

- pentru realizarea impartirii am folosit algoritmul prezentat mai sus si cateva metode suplimentare cum ar fi :

```
-   public int grad(Polinom polinom){
        int grad = -1;
        for(Map.Entry<Integer,Double> entry:
    polinom.getMonoame().entrySet()){
            if(entry.getValue()!=0){
                if( entry.getKey()>grad){
                    grad = entry.getKey();
                }
            }
        }
        return grad;
    }
```

- metoda aceasta returneaza gradului maxim al fiecarui polinom pentru a putea efectua in continuare impartirea

```
-   public Monom cautaMaxim(Polinom polinom){
        int i=0;
        Monom monom = new Monom(0,0);
        for(Map.Entry<Integer,Double >entry :
    polinom.getMonoame().entrySet()){
            if(entry.getValue()!=0){
            if(i==0){
                monom.setCoeficient(entry.getValue());
                monom.setGrad(entry.getKey());
                i++;
            }
            }
        }
        return monom;
    }
```

- aceasta metoda preia primul monom din polinom, adica polinomul cu gradul cel mai mare, polinomul maxim

- aceasta metoda returneaza un ArrayList format din cele doua polinoame rezultate, catul si restul impartirii.

- derivarea polinoamelor:

```
-   public Polinom derivarePolinom(Polinom polinom){

        Polinom rezultat = new Polinom();
        if(polinom.getMonoame().size()==0 )rezultat.addMonom(new
    Monom(0,0));
        for(Map.Entry<Integer, Double> entry:
    polinom.getMonoame().entrySet()){
            if(entry.getKey()>0){
                Monom monom = new
    Monom(entry.getValue()*entry.getKey(), entry.getKey()-1);
                rezultat.addMonom(monom);
            }else{
                rezultat.addMonom(new Monom(0,0));
            }
        }
```

```
        return rezultat;
    }
```

- am parcurs polinomul si am realizat derivarea fiecarui monom in parte dupa regula => (ax^b) => a*bx^(b-1);
- aceasta metoda returneaza polinomul rezultat in urma realizarii operatiei de derivare

- integrarea polinoamelor:

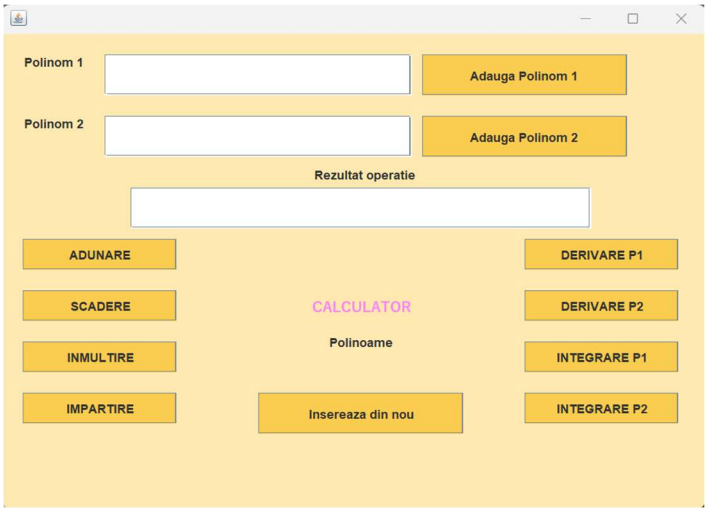
```
-   public Polinom integrarePolinom(Polinom polinom){
        Polinom rezultat = new Polinom();
        if(polinom.getMonoame().size()==0 )rezultat.addMonom(new
    Monom(0,0));
        for(Map.Entry<Integer, Double> entry:
    polinom.getMonoame().entrySet()){
            Monom monom = new Monom(entry.getValue()/
    (entry.getKey()+1), entry.getKey()+1);
            rezultat.addMonom(monom);
        }
        return rezultat;
    }
```

- am parcurs polinomul si am realizat integrarea fiecarul polinom in parte dupa regula de integrare => (ax^b) => (a/(b+1))x^(b+1)
- aceasta metoda returneaza polinomul rezultat in urma efectuarii operatiei de integrare

Interfata Grafica – Clasa Fereastra Principala

- interfata grafica cuprinde urmatoarele elemente:
  - FRAME – ul – acesta este rama in care se afla toate elementele avem nevoie pentru buna functionare a programului. Aceasta fereastra are optiunea de a inchide fereastra la apasarea " X " din dreapta sus.
  - Butoanele – avem un numar de 11 butoane fiecare avand o anumita functionalitate
    - Adauga Polinom 1 – acest buton are rolul de a prelua string-ul din textField-ul corespunzator polinomului 1 si de a-l transforma intr un polinom
    - Adauga Polinom 2 – acest buton are rolul de a prelua string- ul din textField-ul corespunzator polinomului2 si de a-l transforma intr un polinom
    - ADUNARE – acest buton are rolul de a efectua adunarea celor doua polinoame introduse si de a afisa rezultatul in textField-ul corespunzator rezultatului
    - SCADERE – acest buton are rolul de a efectua scaderea celor doua polinoame introduse si de a afisa rezultatul in textField-ul corespunzator rezultatului

- INMULTIRE – acest buton are rolul de a efectua inmultirea celor doua polinoame introduse si de a afisa rezultatul in textField-ul corespunzator rezultatului
- IMPARTIRE – acest buton are rolul de a efectua scaderea celor doua polinoame introduse si de a afisa rezultatul in textField-ul corespunzator rezultatului
- DERIVARE P1 – acest buton are rolul de a efectua derivarea polinomului 1 si de a afisa rezultatul in textField-ul corespunzator rezultatului
- DERIVARE P2 – acest buton are rolul de a efectua derivarea polinomului 2 si de a afisa rezultatul in textField-ul corespunzator rezultatului
- INTEGRARE P1 – acest buton are rolul de a efectua integrarea polinomului 1 si de a afisa rezultatul in textField-ul corespunzator rezultatului
- INTEGRARE P2 – acest buton are rolul de a efectua integrarea polinomului 2 si de a afisa rezultatul in textField-ul corespunzator rezultatului
- Insereaza din nou – acest buton are rolul de a elibera colectia de monoame din fiecare polinom introdus pentru a putea fi introdus unul nou.

- TextField – se regasesc 3 in aceasta interfata:
  - Primul dreptunghi specific label-ului “Polinom 1” in care introducem polinomul 1 in formatului: ax^b(+-)cx^d
  - Al doilea dreptunghi specific label-ului “Polinom 2” in care introducem polinomul 2 in formatului: ax^b(+-)cx^d
  - Al treilea dreptunghi specific label-ului “Rezultat operatie” in care vom scrie rezultatul operatiei selectate in formatului: ax^b(+-)cx^d
- Label – se regasesc 4 in aceasta interfata:
  - “Polinom 1”
  - “Polinom 2”
  - “Rezultat operatie”
  - “CALCULATOR Polinoame”
- 

- Metoda parsePolinom care are ca parametri string-ul din textField si polinomul in care va fi pus string-ul dupa transformarea acestuia.
    - Am folosit expresiile regulate: Pattern, unde am folosit un regex pentru identificare tiparului pe care vrem sa l respecte fiecare polinom si Matcher pentru a vedea daca se regasesc regex-ul in fiecare textField.

```
-   void parsePolinom (String input,Polinom polinom) {
        Pattern p = Pattern.compile("([+-
    ]?\\d*[.]?\\d?)[x](\\^(\\d+))?|([+-]?\\d+)");
        Matcher m = p.matcher(input);
        int grad = 0;
        Double coeficient = 0.0;
        while (m.find()) {
            if (m.group(4) == null) {
                if (m.group(3) == null) {
                    grad = 1;
                } else {
                    grad = Integer.parseInt(m.group(3));
                }
                if (m.group(1) == "") {
                    coeficient = 1.0;
                } else {
                    if (m.group(1).endsWith("-") == true) {
                        coeficient = -1.0;
                    } else {
                        if (m.group(1).endsWith("+") == true) {
                            coeficient = 1.0;
                        } else {
                            coeficient =
    Double.parseDouble(m.group(1));
                        }
                    }
                }
            } else {
                coeficient = Double.parseDouble(m.group(4));
                grad = 0;
            }
            Monom monom = new Monom(coeficient, grad);
            polinom.addMonom(monom);
        }
    }
```

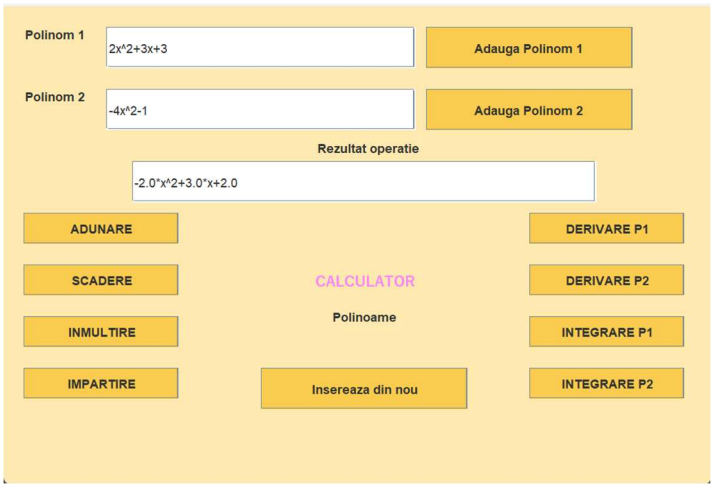
- Metoda check care primeste ca parametru string-ul pe care il introducem in textField si verifica daca polinomul introdus este de o singura variabila

```
-   boolean check(String s) throws Exception{
        if (s == null) {
            return false;
        }
        int len = s.length();
        int k=0;
        for (int i = 0; i < len; i++) {

    if(s.charAt(i)=='x'||s.charAt(i)=='^'||s.charAt(i)=='.'||s.cha
    rAt(i)=='+'||s.charAt(i)=='-
    '||Character.isDigit(s.charAt(i))==true){
```

```
                k++;
            }else{
                throw new Exception("");
            }
        }
        if(k==len){
            return true;
        }

        return false;
    }
```

5. Rezultate

- Rezultatul in urma adunarii a doua polinoame random:



- Rezultatul in urma inmultirii a doua polinoame random:

- Rezultatul impartirii a doua polinoame random :

- 

Clasa OperatiiTest

- Am realizat testarea cu Junit pentru operatiile implementate in clasa Operatii

```
public class OperatiiTest {


    public void initializeazaPolinoame(Polinom polinom1, Polinom polinom2){
        Monom monom1 = new Monom(2,2);
        Monom monom2 = new Monom(3,4);
        polinom1.addMonom(monom1);
        polinom1.addMonom(monom2);
        Monom monom4 = new Monom(3,2);
        Monom monom5 = new Monom(5,3);
        polinom2.addMonom(monom4);
        polinom2.addMonom(monom5);
    }
    @Test
    public void addTest(){
        Polinom polinom1 = new Polinom();
        Polinom polinom2 = new Polinom();
        initializeazaPolinoame(polinom1,polinom2);
        Operatii operatie = new Operatii();
        Polinom rezultat = operatie.adunaPolinoame(polinom1,polinom2);
        assertEquals("3.0*x^4+5.0*x^3+5.0*x^2",rezultat.toString(),"Adunarea
e corecta!");
    }
    @Test
    public void substractTest(){
        Polinom polinom1 = new Polinom();
        Polinom polinom2 = new Polinom();
        initializeazaPolinoame(polinom1,polinom2);
        Operatii operatie = new Operatii();
        Polinom rezultat = operatie.scadePolinoame(polinom1,polinom2);
        assertEquals("3.0*x^4-5.0*x^3-1.0*x^2",rezultat.toString(),"Scaderea
e corecta!");
    }
    @Test
    public void multiplyTest(){
        Polinom polinom1 = new Polinom();
```

```
        Polinom polinom2 = new Polinom();
        initializeazaPolinoame(polinom1,polinom2);
        Operatii operatie = new Operatii();
        Polinom rezultat = operatie.inmultestePolinoame(polinom1,polinom2);

assertEquals("15.0*x^7+9.0*x^6+10.0*x^5+6.0*x^4",rezultat.toString(),"Inmulti
rea e corecta!");
    }
    @Test
    public void divideTest(){
        Polinom polinom1 = new Polinom();
        Polinom polinom2 = new Polinom();
        initializeazaPolinoame(polinom1,polinom2);
        Operatii operatie = new Operatii();
        ArrayList<Polinom> listPolinoms= new ArrayList<Polinom>();
        try {
            listPolinoms = operatie.impartePolinoame(polinom1, polinom2);
        }catch(Exception exp){
            System.out.println("nu e bine introdus!");
        }
        int i=0;
        for(Polinom polinomIterator: listPolinoms){
            if(i==0){
                assertEquals("0.6*x-0.36",polinomIterator.toString(),"Catul e
corect!");
                i++;
            }
            else{
                if(i==1){
                    i++;

assertEquals("0+3.08*x^2",polinomIterator.toString(),"Restul e corect!");
                }
            }

        }
    }
    @Test
    public void derivate1Test(){
        Polinom polinom1 = new Polinom();
        Polinom polinom2 = new Polinom();
        initializeazaPolinoame(polinom1,polinom2);
        Operatii operatie = new Operatii();
        Polinom rezultat = operatie.derivarePolinom(polinom1);
        assertEquals("12.0*x^3+4.0*x",rezultat.toString(),"Derivarea1  e
corecta!");
    }
    @Test
    public void derivate2Test(){
        Polinom polinom1 = new Polinom();
        Polinom polinom2 = new Polinom();
        initializeazaPolinoame(polinom1,polinom2);
        Operatii operatie = new Operatii();
        Polinom rezultat = operatie.derivarePolinom(polinom2);
        assertEquals("15.0*x^2+6.0*x",rezultat.toString(),"Derivarea2  e
corecta!");
    }
```

```
    @Test
    public void integrare1Test(){
        Polinom polinom1 = new Polinom();
        Polinom polinom2 = new Polinom();
        initializeazaPolinoame(polinom1,polinom2);
        Operatii operatie = new Operatii();
        Polinom rezultat = operatie.integrarePolinom(polinom1);

assertEquals("0.6*x^5+0.6666666666666666*x^3",rezultat.toString(),"Integrarea
1  e corecta!");
    }
    @Test
    public void integrare2Test(){
        Polinom polinom1 = new Polinom();
        Polinom polinom2 = new Polinom();
        initializeazaPolinoame(polinom1,polinom2);
        Operatii operatie = new Operatii();
        Polinom rezultat = operatie.integrarePolinom(polinom2);
        assertEquals("1.25*x^4+x^3",rezultat.toString(),"Integrarea2  e
corecta!");
    }
```

6. Concluzii si Dezvoltari ulterioare

- Prin realizarea acestui proiect am exersat folosirea principiilor OOP si am recapitulat folosirea Colectiilor
- Am reusit sa stapanesc mult mai bine polosirea Map-ului
- Am reusit sa construiesc o interfata placuta si usor de utilizat
- Calculatorul ar putea fi dezvoltat adaugand optiunea de adaugare a mai multor polinoame. De asemenea interfata ar putea fi imbunatatita prin adaugarea unor butoane la introducerea elementelor fiecaror polinoame si a nu mai trebui introduse de la tastatura.

7. Bibliografie

- https://dsrl.eu/courses/pt/materials/PT2023_A1_S1.pdf
- https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html
- https://www.jetbrains.com/help/idea/creating-and-opening-forms.html#newForm