DOCUMENTATIE

TEMA *1*

NUME STUDENT: ZUBAȘCU ILEANA
GRUPA: 30224

CUPRINS

1.Obiectivul temei

Obiectivul temei numarul 1 este de a proiecta si implementa un calculator de polinoame cu o interfață grafică în care utilizatorul poate insera polinoame, poate selecta operația matematică care trebuie efectuată și poate vizualiza rezultatul.

Obiective secundare:

-analiza problemei și identificarea cerințelor (cap.2- Analiza problemei, modelare, scenarii, cazuri de utilizare)

-proiectarea calculatorului polinomial (cap.3- Proiectare)

-implementarea calculatorului polinomial(cap.4- Implemenare)

-testarea calculatorului polinomial(cap.5- Rezultate)

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

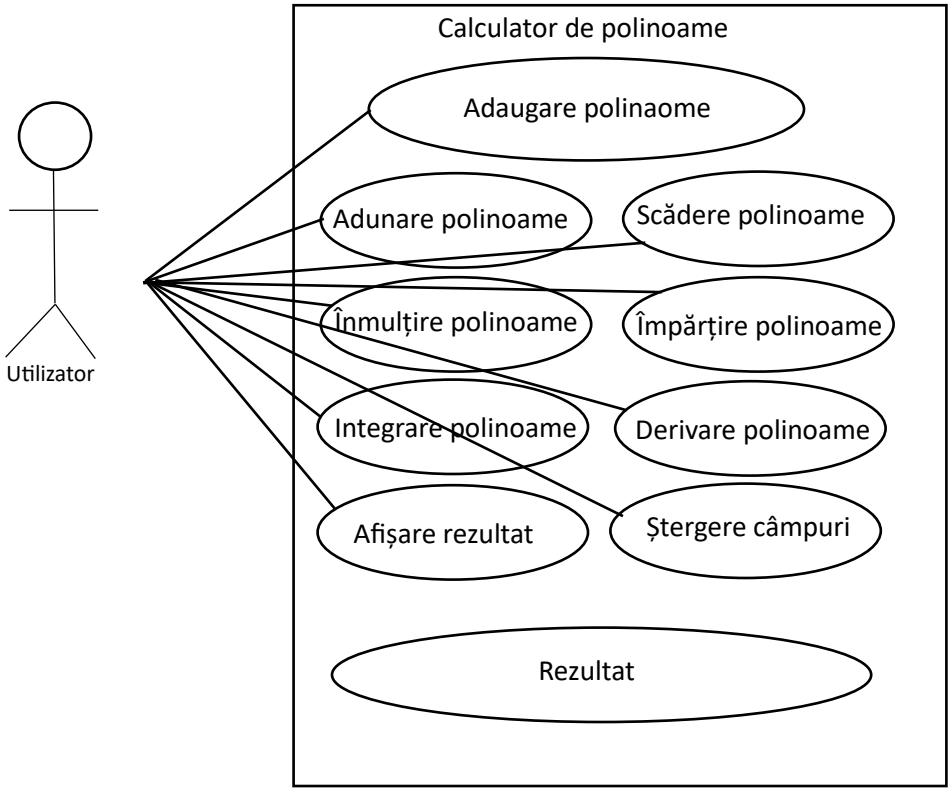
Cerințele funcționale în cadrul acestei teme sunt următoarele:

- Calculatorul de polinoame trebuie să permită utilizatorului să introducă de la tastatură cele două polinoame.
- Calculatorul de polinoame ar trebui să permită utilizatorului să selecteze operația dorită.
- Calculatorul de polinoame ar trebui să afișeze rezultatul la apăsarea unui buton.
- Calculatorul de polinoame ar trebui să șteargă atât polinoamele adăugate, cât și rezultatul la apăsarea unui buton.

Cerințele non funcționale sunt următoarele:

- Calculatorul de polinoame ar trebui să fie intuitiv și ușor de utilizat de către utilizator.
- Calculatorul de polinoame ar trebui să aibă un design sugestiv și atractiv.
- Rezultatul operațiilor ar trebui să fie evidențiat printr-un font colorat care să-l pună în evidență.
- În cazul în care utilizatorul introduce un polinom invalid, se va afișa un mesaj de eroare.

Cazuri de utilizare



Use-case

- Adunare polinoame
  Actorul principal: utilizatorul
  Scenariul de success:
  1. Utliizatorul adaugă în interfața grafică cele două polinoame.
  2. Utilizatorul apasă pe operația de adunare (“add”).
  3. Utilizatorul apasă pe butonul de calcul (“=”).
  4. Calculatorul efectuează adunarea și afișează rezultatul.

- Scădere polinoame
  Actorul principal: utilizatorul
  Scenariul de success:
  1. Utilizatorul adaugă în interfața grafică cele două polinoame.
  2. Utilizatorul apasă pe operația de scădere (“substract”).
  3. Utilizatorul apasă pe butonul de calcul (“=”).
  4. Calculatorul efectuează scăderea și afișează rezultatul.

- Înmulțire polinoame
  Actorul principal: utilizatorul
  Scenariul de success:
  1. Utilizatorul adaugă în interfața grafică cele două polinoame.
  2. Utilizatorul apasă pe operația de înmulțire (“multiplicate”).
  3. Utilizatorul apasă pe butonul de calcul (“=”).
  4. Calculatorul efectuează înmulțirea și afișează rezultatul.

- Împărțire polinoame
  Actorul principal: utilizatorul
  Scenariul de success:
  1. Utilizatorul adaugă în interfața grafică cele două polinoame.
  2. Utilizatorul apasă pe operația de împărțire (“divide”).
  3. Utilizatorul apasă pe butonul de calcul (“=”).
  4. Calculatorul efectuează împărțirea și afișează rezultatul.

- Integrare polinom
  Actorul principal: utilizatorul
  Scenariul de success:
  1. Utilizatorul adaugă în interfața grafică câmpul primului polinom(“Polynom 1”).
  2. Utilizatorul apasă pe operația de integrare (“integration”).
  3. Utilizatorul apasă pe butonul de calcul (“=”).
  4. Calculatorul efectuează integrarea și afișează rezultatul.

- Derivare polinom
  Actorul principal: utilizatorul
  Scenariul de success:
  1. Utilizatorul adaugă în interfața grafică câmpul primului polinom(“Polynom 1”).
  2. Utilizatorul apasă pe operația de derivare (“derivation”).
  3. Utilizatorul apasă pe butonul de calcul (“=”).
  4. Calculatorul efectuează calculul de derivare și afișează rezultatul.

Secvență alternativă: polinoamele sunt introduse greșit(**polinoamele trebuie să conțină monomi de forma “x^putere” obligatoriu, coeficentul este opțional**)

-utilizatorul a introdus polinoame incorecte

-utilizatorul apasă pe butonul de calcul

-apare un mesaj de eroare “*Invalid polynom*”

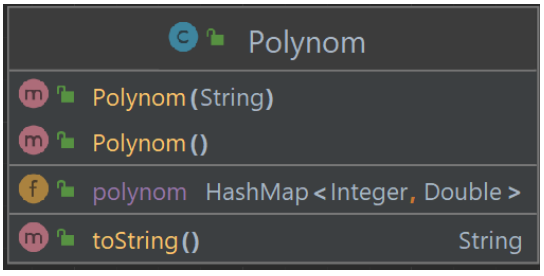
-pentru a șterge polinoamele introduse, utilizatorul poate să apese butonul de ștergere

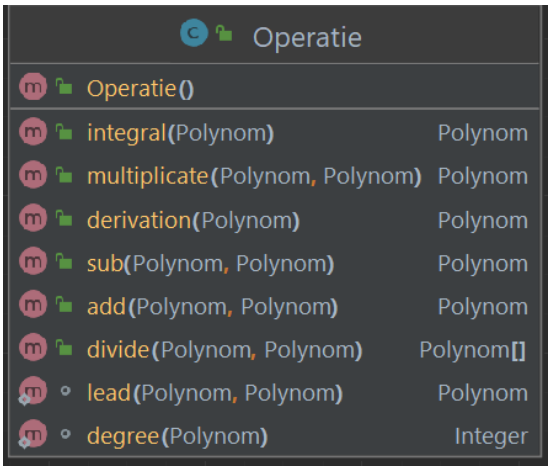
-scenariul revine la pasul 1

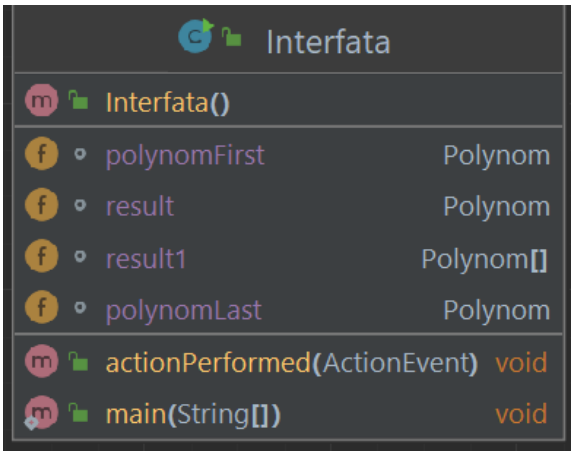
3.Proiectare

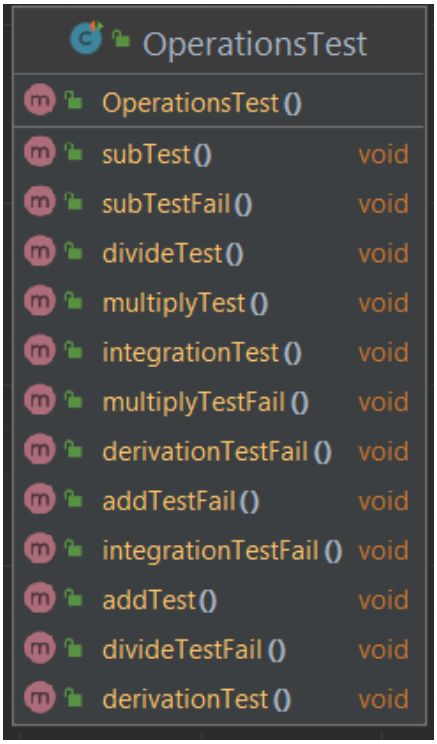
Pentru a stoca polinomul citit initial ca String, am folosit HashMap<Integer,Double> în care am adăugat fiecare monom (Integer pentru gradul monomului și Double pentru coeficient)folosind metoda “split”.Pentru reținerea rezulatului la operația de împărțirea am folosit structura de date de tip vector de “Polynom”.Pentru dezvoltarea interfeței grafice am folosit Java Swing.

Diagrame UML de clase și pachete









4.Implementare

Clasa Polynom:

-se află în pachetul *Data*

-este clasa în care se desparte polinomul citit ca String folosind *HashMap<Integer,Double>*

-conține doi constructori, primul având ca parametru de intare un *String* care stochează coeficientul și puterea fiecărui monom (*Integer* pentru putere și *Double* pentru coeficient).Regula principală în scrierea polinomului este de a specifica obligatoriu puterea fiecărui monom.Am folosit metoda *split* pentru a despărți string-ul în monoame de același timp, după care am extras primul număr (dacă acesta există) ca și coeficient și al doilea număr ca și putere.Al doilea constructor este gol, folosit pentru metodele operațiilor.

-conține o singură metodă: toString-returnează un *String* care reprezintă polinomul scris sub formă matematică. În această metodă am folosit *StringBuilder* pentru a construi și a concatena monomii stocați în *HashMap*.

Clasa Operatie

-se află în pachetul *Logic*

-conține un constructor gol

-în această clasă sunt implementate toate metodele care efectuează calculele pe polinomi:

- metoda *add* returnează un obiect de tip *Polynom* care reprezintă adunarea a două obiecte de același tip. Prima data se reține primul polinom, după care se parcurge al doilea polinom și se adună monomii de același grad.În cazul în care rezultatul adunării este 0, monomul este șters din rezultat.Codul metodei *add*:

```
4 usages
public Polynom add(Polynom a, Polynom b) {

    Polynom result = new Polynom();
    for (Map.Entry<Integer, Double> entryA : a.polynom.entrySet()) {
        result.polynom.put(entryA.getKey(), entryA.getValue());
    }

    for (Map.Entry<Integer, Double> entryB : b.polynom.entrySet()) {
        if (result.polynom.containsKey(entryB.getKey())) {
            Double s=entryB.getValue() + result.polynom.get(entryB.getKey());
            s=Math.round(s*100.)/100.0;
            if(s.compareTo(0.0)==0)
              result.polynom.remove(entryB.getKey());
            else
                result.polynom.replace(entryB.getKey(),s);

        } else
            result.polynom.put(entryB.getKey(), entryB.getValue());


    }

    return result;
}
```

- metoda *sub* returnează un obiect de tip *Polynom* care reprezintă scăderea a două obiecte de același tip. Prima dată se reține primul polinom, după care se parcurge al doilea polinom și se scad monomii de același grad. Monomii cu grad unic din al doilea polinom sunr reținuți cu coeficientul opus. În cazul în care rezultatul scăderii este 0, monomul este șters din rezultat.Codul metodei *sub*:

```
4 usages
public Polynom sub(Polynom a, Polynom b) {
    Polynom result = new Polynom();
    for (Map.Entry<Integer, Double> entryA : a.polynom.entrySet()) {
        result.polynom.put(entryA.getKey(), entryA.getValue());
    }

    for (Map.Entry<Integer, Double> entryB : b.polynom.entrySet()) {
        if (result.polynom.containsKey(entryB.getKey())) {
            Double s = result.polynom.get(entryB.getKey()) - entryB.getValue();
            s = Math.round(s * 100) / 100.0;
            if(s.compareTo(0.0)==0)
              result.polynom.remove(entryB.getKey());
            else
                result.polynom.replace(entryB.getKey(),s);
        } else
            result.polynom.put(entryB.getKey(), -entryB.getValue());


    }

    return result;
}
```

- metoda *multiplicate* returnează un obiect de tip *Polynom* care reprezintă înmulțirea a două obiecte de același tip. Se parcurg cele două polinoame simultan, se adună puterile celor două monoame și se înmulțesc coeficienții.În cazul în care gradul polinomului rezultat conține deja un monom de același grad, se adună coeficienții, după care se stohează rezultatul în noul polinom .Codul metodei *multiplicate*:

```
public Polynom multiplicate(Polynom a, Polynom b) {
    Polynom result = new Polynom();
    a.polynom.forEach((keyA, valueA) ->
    {
        b.polynom.forEach((keyB, valueB) ->
        {
            Integer keyResult = keyA + keyB;
            Double valueResult = valueA * valueB;
            if (result.polynom.containsKey(keyResult)) {
                result.polynom.put(keyResult, result.polynom.get(keyResult) + valueResult);
            }
            else
                result.polynom.put(keyResult,valueResult);


        });
    });

    return result;

}
```

- metoda *divide* returnează un vector de tip *Polynom* în care se află rezultatul împărțirii a două polinoaome sub formă de “cât”(q) și “rest”(r). Metoda a fost implementată după pseudocodul:

```
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)
```

funcția *degree* calculează gradul polinomului, iar funcția *lead* returnează monomul rezultat în urma împărțirii între doi monomi. Codul metodei *divide*:

```
public Polynom[] divide(Polynom n, Polynom d)
{
    Polynom q=new Polynom();
    Polynom r=n;
    while(r.polynom.size()!=0 && degree(r)>=degree(d))
    {
        Polynom t=lead(r,d);
        q=add(q,t);
        r=sub(r,multiplicate(t,d));

    }
    Polynom[] result={q,r};

    return result;
}
```

- Metoda *derivation* returnează un obiect de tip *Polynom* care reprezintă rezultatul derivării polinomului descris ca parametru de intrare. Metoda scade gradul ficărui monom și înmulțește fiecare coeficient cu gradul monomului din care face parte.Codul metodei *derivation*:

```
public Polynom derivation(Polynom a) {
    Polynom result = new Polynom();
    for (Map.Entry<Integer, Double> entryA : a.polynom.entrySet()) {
        result.polynom.put(entryA.getKey() - 1, entryA.getValue() * entryA.getKey());
    }

    return result;
}
```

- Metoda *integral* returnează un obiect de tip *Polynom* care reprezintă rezultatul integrăării polinomului descris ca parametru de intrare. Metoda adună gradul ficărui monom cu 1 și înmulțește fiecare coeficient cu gradul monomului+1 din care face parte.Codul metodei *integral*:

```java
public Polynom integral(Polynom a) {
    Polynom result = new Polynom();
    for (Map.Entry<Integer, Double> entryA : a.polynom.entrySet()) {
        Double s = entryA.getValue() / (entryA.getKey() + 1);
        s = Math.round(s * 100) / 100.0;
        result.polynom.put(entryA.getKey() + 1, s);
    }

    return result;
}
```

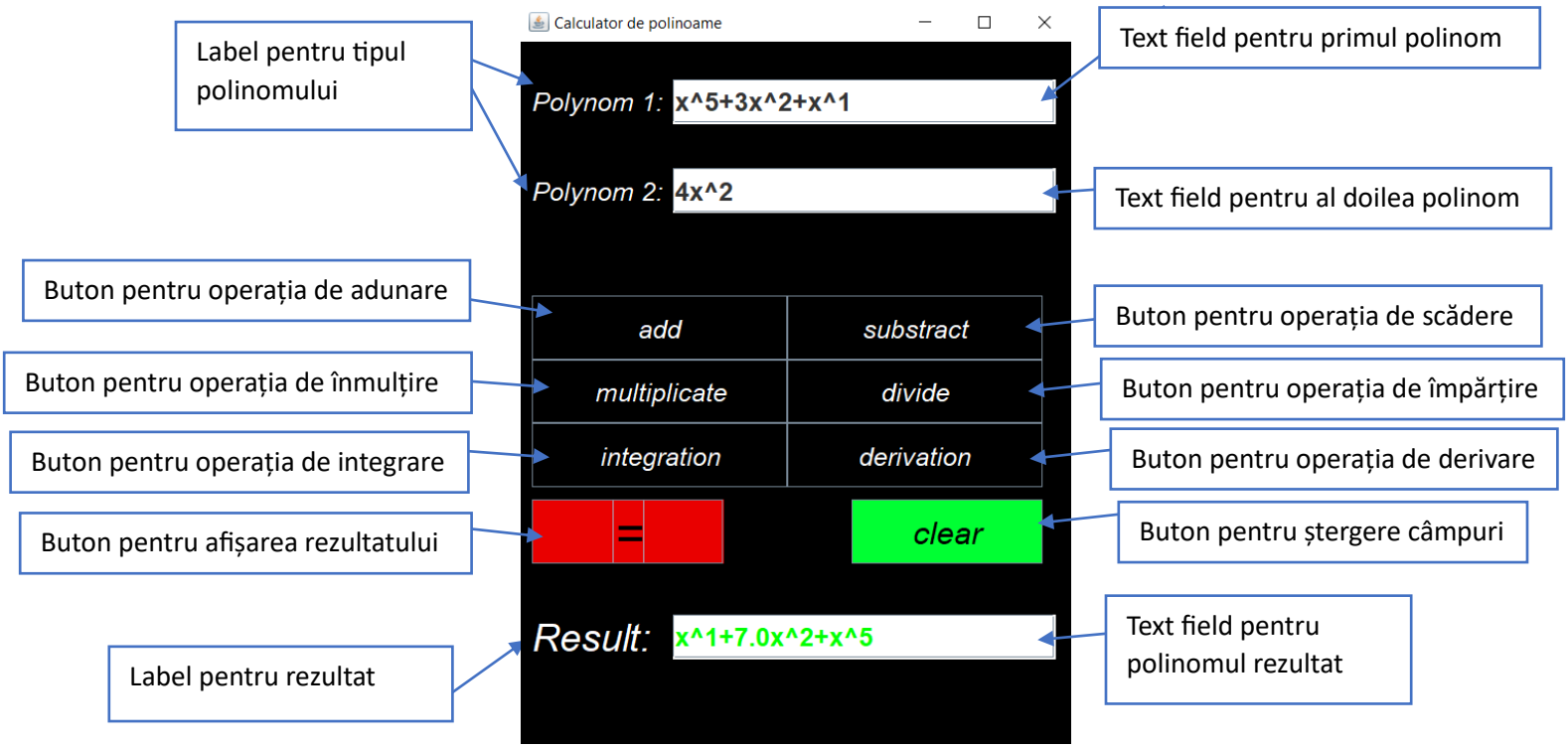
Clasa Interfata

-se află în pachetul GUI

-în această clasă se crează interfața grafică a calculatorului de polinoame folosing Java Swing. Pentru implementare am folosit obiectele de tip JFrame (cadrul principal), JTextField (field-uri pentru scrierea polinomelor și afișarea rezultatului), JLabel (folosit în descrierea text field-urilor), JButton (butoane pentru operații, afișare rezultat și clear).

-conține metoda main care lansează interfața grafică

Interfața grafică:

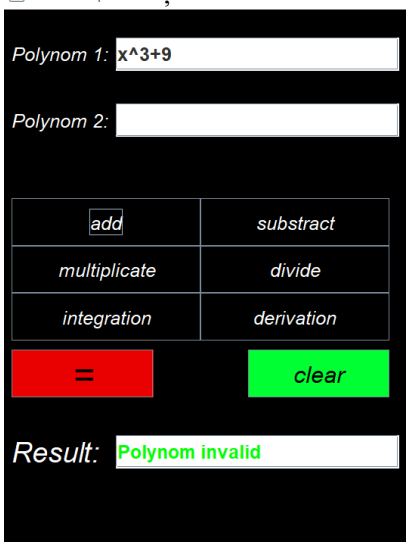


-constructorul clasei implementaează interfața utilizator. Prima data am definit componentele enumerate mai sus, după care am folosit metoda *addActionListener* pentru fiecare buton.

Codul pentru butonul *add*(operația de adunare) -celălalte butoane pentru operații sunt similare:

-se citesc cele două polinoame din text filed-uri, după care se verifică dacă acestea sunt valide. Dacă polinoamele n-au fost scrise corect, atunci se afișează un mesaj de eroare (“*Polynom invalid*”) în text field-ul rezultatului, în caz contrar se face operația de adunare și se stochează în obiectul de tip *Polynom* **result**.

```
add.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        polynomFirst= new Polynom(polynomA.getText());
        polynomLast= new Polynom(polynomB.getText());
        if(polynomFirst.polynom.isEmpty() || polynomLast.polynom.isEmpty())
            polynomR.setText("Polynom invalid");
        else{
        Operatie o= new Operatie();
        result=o.add(polynomFirst,polynomLast);
    }}
});
```



Codul pentru butonul *equal*(afișarea rezultatului):

-în cazul în care rezultatul nu există, se afișează *0* în text filed-ul rezultatului, în caz contrat se convertește rezultatul la *String*, după care se afișează la *Result*.

```
equal.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

            if (result.polynom.isEmpty())
                polynomR.setText("0");
            else
                polynomR.setText(result.toString());
        }

});
```

Codul pentru butonul *clear* (ștergere câmpuri):

-setează toate text filed-urile la caracterul *“”*.

```
clear.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        polynomR.setText("");
        polynomA.setText("");
        polynomB.setText("");

    }
});
```

5.Rezultate

Testarea operațiilor a fost efectuată în calsa *OpeartionsTest* din pachetul *TestOperation* prin intermediul testării unitare *JUnit* făcând configurări la *Maven* în *pom.xml*. Am adăugat la fiecare operație două metode de testare în care una este corectă și una greșită(în total 6 teste corecte și 6 teste greșite):

```
@Test
public void addTest(){
    Operatie o=new Operatie();
    Polynom polynom= o.add(new Polynom( p: "x^1"), new Polynom( p: "x^2"));
    String actualResult=polynom.toString();
    assertEquals( expected: "x^1+x^2",actualResult);
}
@Test
public void addTestFail(){
    Operatie o=new Operatie();
    Polynom polynom= o.add(new Polynom( p: "x^1"), new Polynom( p: "x^2"));
    String actualResult=polynom.toString();
    assertEquals( expected: "x^1",actualResult);
```

6.Concluzii

În concluzie, tema 1 mi-a dezvoltat tehnica de scriere a codului în Java, mi-a îmbunătățit modul în care îmi organizez codul și modul în care analizez cerințele și problema prezentată astfel încât să implementez un cod cât mai efficient și correct.

Posibile dezvoltări ulterioare ar putea fi: să nu fie obligatorie putearea fiecărui monom, despărțirea în monoame să fie îmbunătățită cu ajutorul unor funcții și ,în urma împărțirii, rezultatul să apară doar după apăsarea butonului de afișare.

7.Bibliografie

https://dsrl.eu/courses/pt/

https://www.w3schools.com/java/java_hashmap.asp

https://www.javatpoint.com/java-swing

https://www.symbolab.com/solver/polynomial-equation-calculator

https://www.geeksforgeeks.org/how-to-divide-polynomials/