DOCUMENTAȚIE

TEMA 1

NUME STUDENT: Marcu Ariana-Mălina
GRUPA: 30222

CUPRINS

1. Obiectivul temei

Obiectivul temei este de a implementa in JAVA un calculator ce realizeaza diferite operatii (adunare, scadere, inmultire, impartire, derivare, integrare) pe polinoame. Cu ajutorul unei interfete grafice, utilizatorul poate sa introduca propriile exemple de polinoame, sa aleaga operatiile pe care doreste sa le efectueze si sa vada rezultatele.

Obiectivele secundare ale acestei teme vor fi prezentate mai detaliat in capitolul 4 al acestei documentatii si sunt reprezentate de:

- Crearea clasei Monomial, care sa stocheze partile principale ale unui monom: coeficientul si puterea.
- Crearea clasei Polynomial care, in aceeasi maniera, sa detina gradul polinomului si un TreeMap de monoame.
- Implementarea celor mai importante operatii, in cazul meu tot in clasa Polynomial, pentru a-mi fi mai usor sa accesez direct o anumita operatie pe un anumit polinom dat, conform Programarii Orientate pe Obiect.
- Clasa Metode, care sa contina ca principale metode citirea polinomului dat ca String si parsarea acestuia, dupa citire, din Map in monoame.
- Organizarea calculatorului folosind interfata grafica GUI.
- Testarea aplicatiei folosind testarea unitara JUnit in clasa numita PolynomialTest.

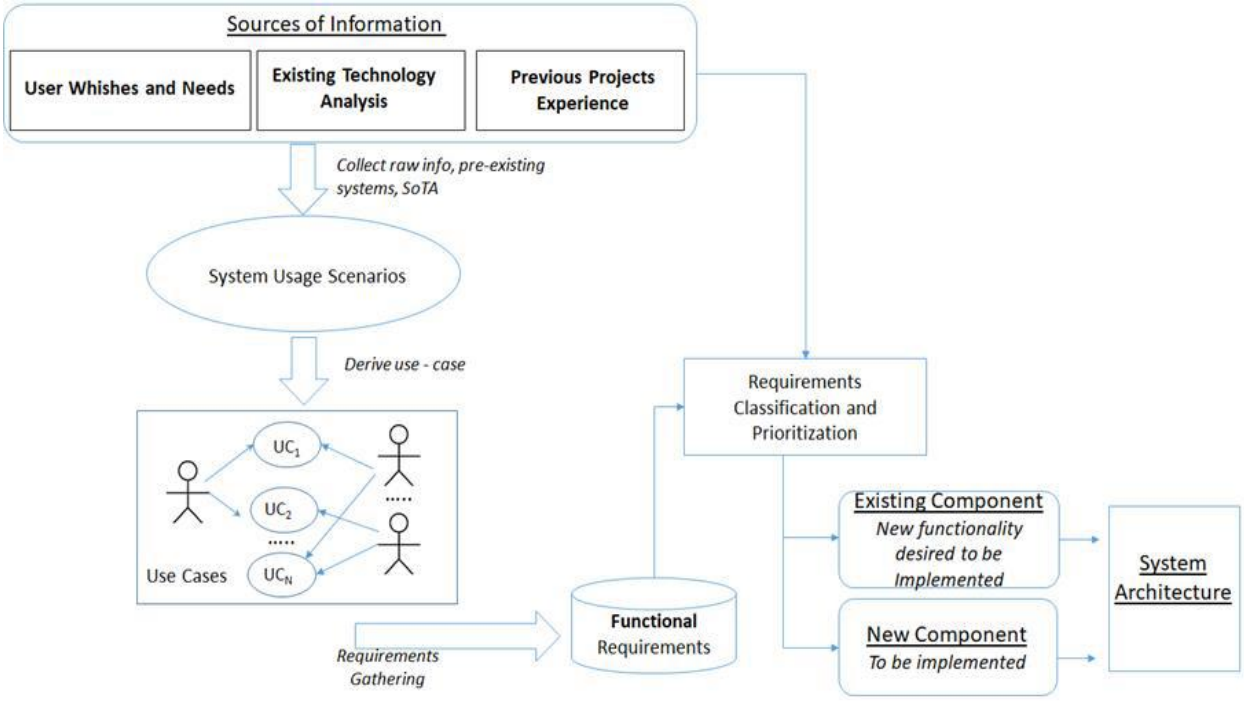
2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Analiza oricarei probleme ar trebui sa inceapa cu identificarea datelor de intrare si a datelor de iesire. Cerintele functionale ale unui calculator de polinoame in Java includ capacitatea de a:

- ✓ introduce polinoame prin intermediul tastaturii
- ✓ aduna, scadea, inmulti, deriva sau integra (in cazul meu) polinoame
- ✓ afisa polinoame si rezultatele operatiilor dintre ele

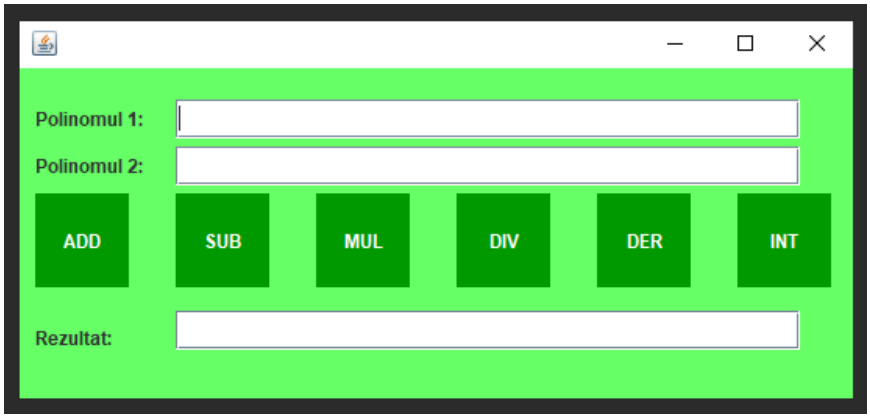
Cerintele non functionale:

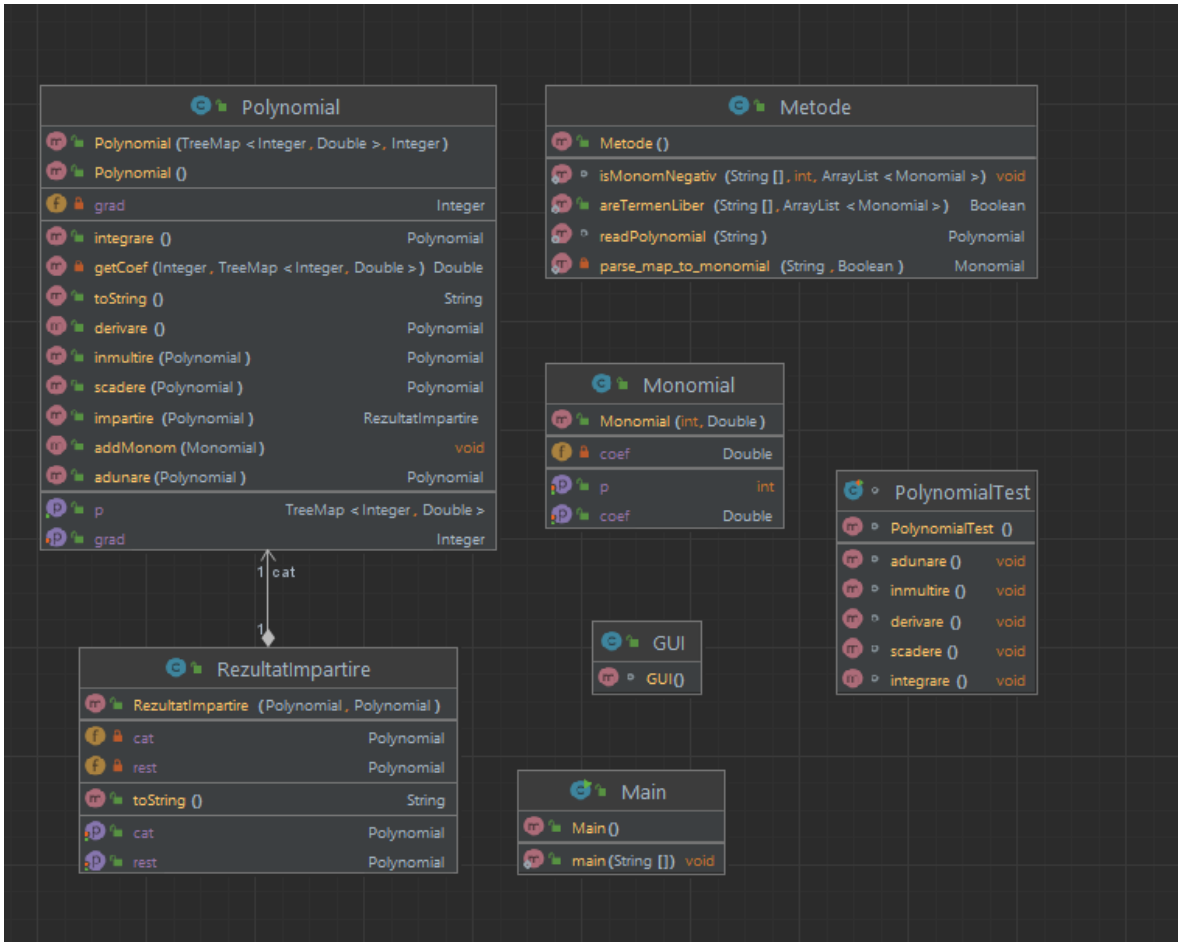
- ✓ performanta buna si viteza a operatiilor
- ✓ interfata cu utilizatorul usor de utilizat si avantajoasa pentru implementare
- ✓ capacitatea de a gestiona polinoame de grade mari fara a reduce performanta
- ✓ siguranta, extensibilitate, evitare erori si exceptii



Scenariul principal de utilizare al unui calculator de polinoame ar fi cazul in care avem de a face cu polinoame de grad mare, pe care, incercand sa le calculam pe foaie, ar necesita foarte mult timp, sau pur si simplu pentru a calcula o operatie dorita intre doua polinoame. Operatia poate fi selectata prin apasarea unor butoane specifice, intuitive, iar rezultatul se va afisa imediat in interfata.

3. Proiectare





Mai sus am prezentat interfata grafica a proiectului, pe care o realizez in intregime in clasa GUI, care se poate vedea in cea de a doua poza - diagrama UML. Folosindu-ma de Java Swing, am creat pe rand fiecare buton si TextField, urmand ca apoi sa atribui fiecarui buton cu ActionListener, functionalitatea specifica. Bazandu-ma pe POO (Obiectul = Polinomul), clasele si structurile de date pe care le-am folosit in proiectare sunt:

- ✓ Monomial – clasa importanta a proiectului, care defineste de fapt structura de baza a unui monom, cu care apoi sa se poate lucra in polinom. Un monom este de fapt format din coeficient, pe care l-am setat Double doar pentru a putea realiza mai tarziu integrarea si eventual impartirea, baza, care este x la toate monoamele deci nu e nevoie sa fie stocata, si in final puterea/exponentul, Integer.
- ✓ Polynomial – clasa fundamentala a proiectului. Aici va fi nevoie cu siguranta de un tablou de monoame, stocat intr-un TreeMap de Integer(cheia unica – puterea) si Double(valoarea – coeficientul), si pentru a nu se pierde gradul maxim al unui polinom de-a lungul operatiilor, mai avem nevoie si de un Integer grad. Totodata aici sunt implementate toate operatiile.
- ✓ Metode – clasa in care realizez citirea unui polinom pe baza Stringului din care este format, in care vad si daca fiecare monom este sau nu negativ si daca polinomul are sau nu termen liber, urmand ca polinomul rezultat sa il parsez in monoame cu care realizez de fapt, in functie de putere, adunarile, scaderile, etc.
- ✓ RezultatImpartire – clasa in care ar fi trebuit sa stochez catul si restul din impartirea polinoamelor
- ✓ GUI – interfata grafica
- ✓ Main – in care tot ce fac este sa apelez GUI

4. Implementare

1. *CLASA GUI (Graphical User Interface)*

Contine un JFrame pe care se construieste intreaga interfata. Avem deci o singura zona principala, cea de control, in care se regasesc campurile si butoanele prin care utilizatorul introduce datele si poate sa vada primele rezultate concrete:

```
JPanel panel = new JPanel();
JFrame frame = new JFrame();
frame.setSize(550, 250);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.add(panel);
panel.setLayout(null);
frame.setVisible(true);
```

Felul in care creez fiecare TextField este urmatorul:

```
JLabel label1 = new JLabel("Polinomul 1: ");
label1.setBounds(10, 20, 80, 25);
panel.add(label1);

JTextField Polinom1 = new JTextField();
Polinom1.setBounds(100, 20, 400, 25);
panel.add(Polinom1);
```

Se poate observa structura de baza a unei zone de afisare a datelor despre polinoame, iar mai jos modul implicit in care realizez fiecare buton si il adaug in panel (ADD – adunare, SUB – scadere, MUL – inmultire, DIV – impartire, DER – derivare, INT – integrare):

```
JButton buton1 = new JButton("ADD");
buton1.setBounds(10, 80, 60, 60);
panel.add(buton1);
```

Pentru realizarea fiecarei operatii in concordanta cu apasarea butonului, ma folosesc de ActionEvent si ActionListener. Apoi apelez polinoamele care se vor da de la tastatura, create mai sus in TextField, le citesc, si apelez operatia intre ele, al carei rezultat vine pus in Polinom3, cu metoda toString din clasa Polynomial. In final, am ales un verde pastel pentru a colora interfata si am setat sa lipseasca border-ul in jurul butoanelor.

```
buton1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String p1String = Polinom1.getText();
        String p2String = Polinom2.getText();
        Polynomial p1 = Metode.readPolynomial(p1String);
        Polynomial p2 = Metode.readPolynomial(p2String);
        Polynomial p3 = p1.adunare(p2);
        Polinom3.setText(p3.toString());
    }
});
```

2. *CLASA POLYNOMIAL*

Permite crearea efectiva a polinoamelor, ca si colectii de monoame: private TreeMap<Integer,Double> polinom si private Integer grad. Folosesc constructorul Polynomial, precum si settere si gettere pentru a accesa mai usor. Mai folosesc si metoda void de addMonom, pe care o folosesc mai tarziu la operatii. In continuare, implementarea incepe cu metoda de adunare a polinoamelor, la care trimit ca parametru un polinom si totodata returnez un polinom. Ma folosesc de un TreeMap pentru rezultat pentru a lucra mai usor direct pe vector si mai ales de metoda private Double getCoef(Integer grad, TreeMap<Integer, Double> polinom). Scopul ei este sa caute coeficientul asociat cu gradul dat in polinom si returneaza valoarea gasita. Mai exact, parcurge fiecare pereche (g, c) din TreeMap si verifica daca g este egal cu gradul dat. AtomicReference-ul imi permite setarea valorii de returnat direct in interiorul blocului forEach, deoarece e necesara o variabila final pentru a fi utilizata in blocul lambda din bucla.

```
public void setGrad(Integer grad) {
    this.grad = grad;
}
public Polynomial(){
    this.polinom = new TreeMap<Integer, Double>();
};
public Polynomial (TreeMap <Integer, Double> map, Integer gradul){
    this.polinom = map;
    this.grad = gradul;
}
public TreeMap<Integer, Double> getP() {
    return polinom;
}
public void addMonom(Monomial monom){
    polinom.put(monom.getP(), monom.getCoef());
}
```

Analog adunarii, realizez si scaderea si inmultirea. La inmultire, voi avea nevoie de for in for, deoarece acolo se realizeaza atat operatii de replace cat si de put. Coeficientii se inmultesc, iar gradele se aduna. In ceea ce priveste derivarea si integrarea, acestea se vor realiza bazat pe "Polinomul 1" dat de la tastatura. In implementare, m-am folosit de formulele matematice: (x^n)' = nx^(n-1) si ∫x^n dx = x^(n+1) / n+1.

In incheiere, pentru ca afisarea rezultatului sa fie in conformitate cu formatul unui polinom, m-am folosit de metoda toString( ), in care imi vine in ajutor StringBuilder, o clasa mutabila, care permite modificarea sirurilor de caractere.

3. *CLASA METODE*

Cu aceasta clasa incepe de fapt implementarea proiectului. Initial, am inceput cu niste metode ajutatoare numite areTermenLiber si isMonomNegativ, cu nume sugestiv. Metodele principale de aici sunt de fapt `static Polynomial readPolynomial(String polinom)` si `private static Monomial parse_map_to_monomial(String s, Boolean is_negativ)`. Prima metoda realizeaza o impartire a monoamelor dupa semnul lor regasit in Stringul dat ca parametru (am grija si de cazul in care polinomul dat al doilea are cumva gradul mai mare decat primul, si sa nu fie ratat) si cea de a doua parseaza intreg TreeMap-ul in monoame, pe 3 cazuri: cazul in care necunoscuta x are putere (adica ^), cazul in care x-ul are puterea 1, si cazul in care avem de a face doar cu un termen liber.

4. *CLASA MONOMIAL*

Clasa aceasta este cea mai scurta, urmarind doar structura obiectului de tip monom, ce are ca parti principale coeficientul si puterea, pe care sa le putem accesa din alte clase cu settere si gettere:

```
private Integer putere;
private Double coef;

public int getP() {
    return putere;
}
public void setP(int p) {
    this.putere = putere;
}

public Double getCoef() {
    return coef;
}

public Monomial(int putere, Double coef) {
    this.putere = putere;
    this.coef = coef;
}
```

5. Rezultate

Pentru o testare mai eficienta, m-am folosit de JUnit Testing. Astfel, o alta clasa regasita este PolynomialTest, in care realizez operatiile necesare. Dupa o multime de teste, si incercand sa acopar toate 'corner case – urile', toate operatiile par sa functioneze destul de bine, indiferent de ordinea (gradelor), in care va introduce utilizatorul polinomul, furnizand rezultate corecte din punct de vedere matematic. Daca polinoamele nu sunt corect scrise, va aparea o exceptie si daca se vor reintroduce corect, programul va functiona fara probleme. Pentru o verificare mai amanuntita, am dat spre exemplu, pentru inmultire, pana si un grad de 7:

```
@Test
void inmultire() {
    Polynomial p1 = Metode.readPolynomial("x^3+7x^2-9x-100");
    Polynomial p2 = Metode.readPolynomial("x^2+5x-7x^7");
    Polynomial p4 = p1.inmultire(p2);
    assertEquals(p4.toString(), "-7,0x^10 - 49,0x^9 + 63,0x^8 + 700,0x^7 + x^5
+ 12,0x^4 + 26,0x^3 - 145,0x^2 - 500,0x");
}
```

Dupa rularea “PolynomialTest”, se va vedea ca la inmultire, rezultatul calculat coincide cu cel dat prin assert. Am dat de asemenea, pentru derivare si adunare, cate un exemplu gresit, pentru a arata functionarea.

6. Concluzii

Prin intermediul acestei teme, consider ca am avut sansa sa imi dezvolt destul de bine capacitatile de a scrie singura cod functional Java, ba chiar mai mult, am avut oportunitatea sa inteleg Unit Testing ul, sa fac o interfata de la zero si chiar sa inteleg mult mai bine structurile de tip Map sau chiar bucla forEach. Chiar cred ca abordarea pe care am ales-o, se afla in directia buna in ceea ce priveste tehnicile de programare orientata pe obiecte, insa mereu se pot gasi imbunatatiri. O posibila dezvoltare ulterioara ar fi implementarea operatiei de impartire a polinoamelor (probabil cu ajutorul schemei lui Horner), organizarea mai atenta a codului sursa in pachete si poate chiar folosirea Regex (regular expressions si pattern matching) pentru extragerea coeficientilor si pentru a beneficia de un cod mai eficient si mai usor de inteles.

7. Bibliografie

1. Bruce Eckel, Thinking in Java (4th Edition), Publisher: Prentice Hall PTRUpper Saddle River, NJUnited States, ISBN:978-0-13-187248-6 Published:01 December 2005.
2. What are Java classes? - www.tutorialspoint.com
3. https://dsrl.eu/courses/pt/materials/lectures/
4. Free Online Diagram Editor
5. How to Use Borders (The Java™ Tutorials > Creating a GUI With Swing > Using Swing Components) (oracle.com)
6. RegExr: Learn, Build, & Test RegEx