# Tema 1

## I. INTRODUCERE

În această temă se va implementa şi analiza un program de gestiune a operaţiilor aritmetice pe doua polinoame. Se vor analiza elemente precum:

1. Interfaţa grafică;
2. Reprezentarea unui Monom în java;
3. Reprezentarea unui Polinom ca listă de monoame;
4. Operaţiile implementate pe polinoame;
5. Concluzie şi analiza a rezolvării problemei;

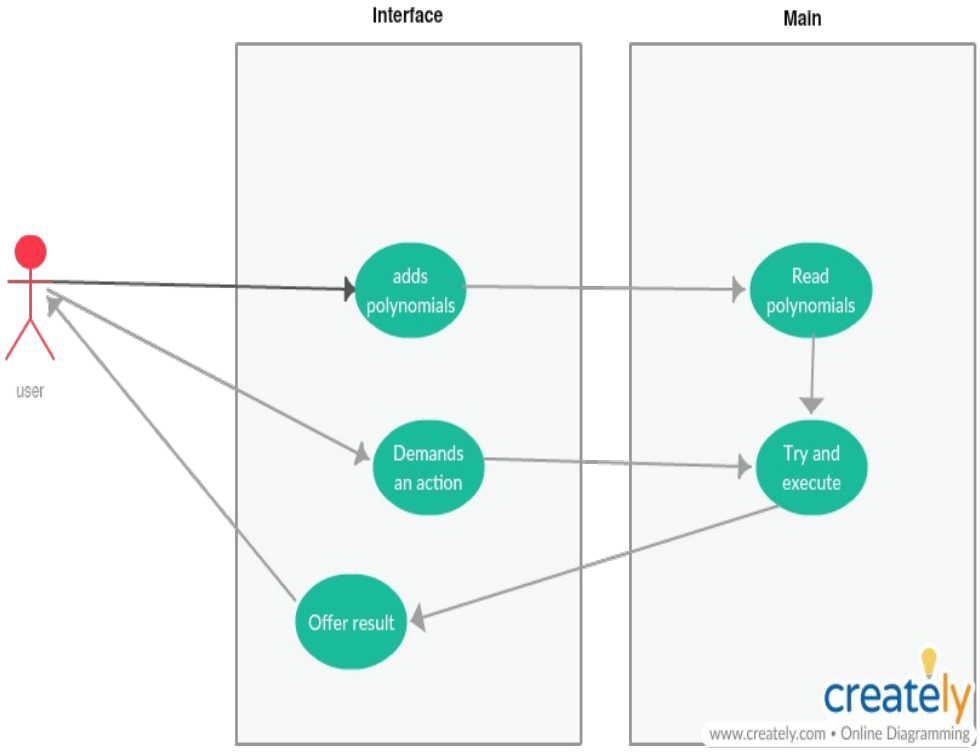
## II. ANALIZA PROBLEIMEI ŞI A CAZURILOR POSIBILE

Textul problemei: „Propuneti, proiectati si implementati un sistem de procesare a polinoamelor de o singura variabila cu coeficienti intregi. “

Pentru a realiza această aplicaţie va fi nevoie de o înţelegere a conceptelor legate de polinoame şi a nevoilor ce pot exista legate de acestea.

În principal, operaţiile pe polinoame sunt adunarea, scăderea, înmulţirea, împărţirea, derivarea şi integrarea. Înafara de împărţire, operaţiile rămase sunt relativ facile în implementare.

Use-case-ul general reprezintă scrierea în aplicaţie a unor polinoame şi executarea unor operaţii:



Utilizatorul nu are alte posibilităţi decât să introducă date şi să ceară rezultate, astfel că erorile pot interveni doar la furnizarea greşită a polinoamelor. Astfel, se va impune un mod de scriere al polinoamelor astfel c1*x^e1 +/-c2*x^e2 +/-...

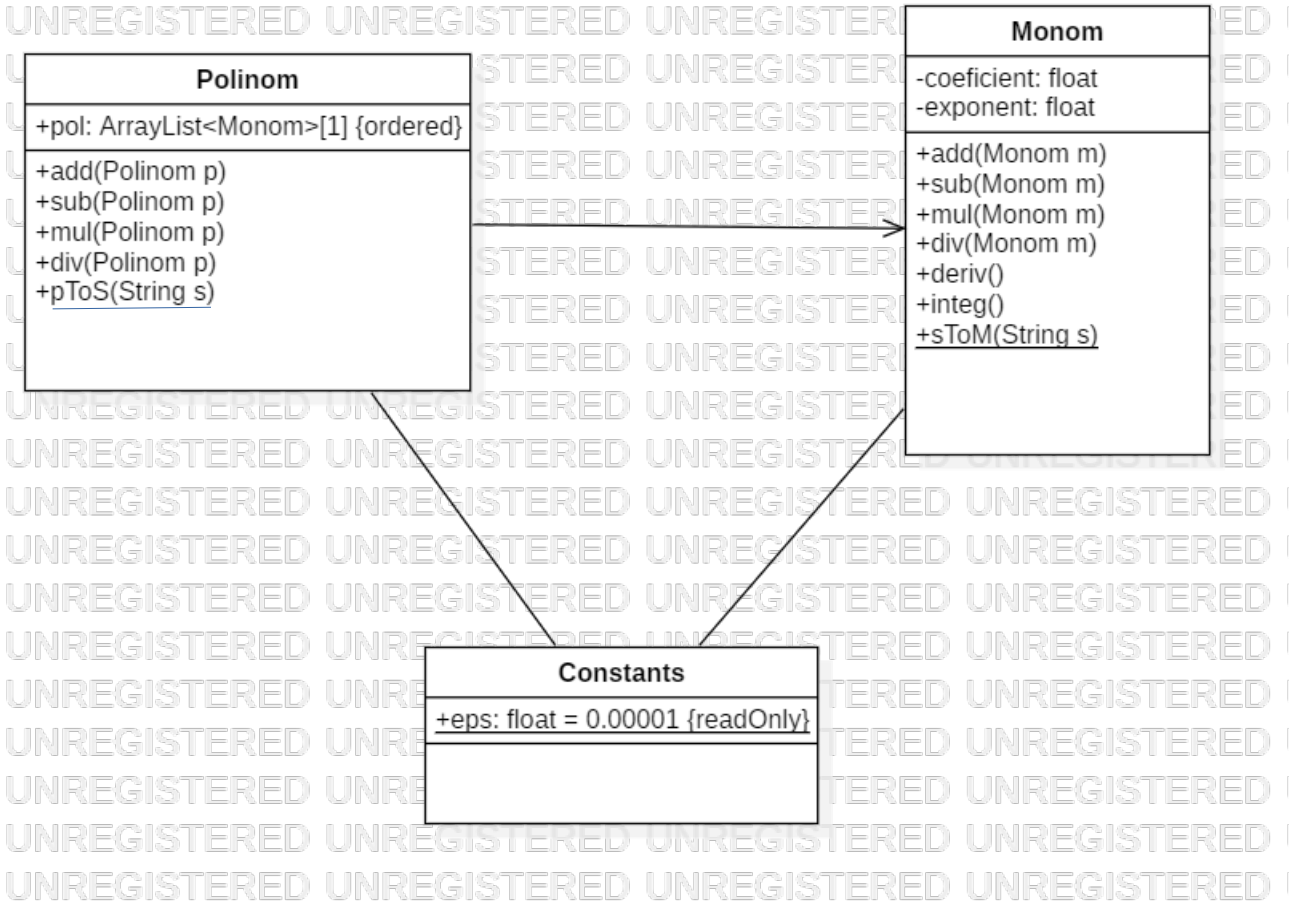
Exemplu : Din versiunea naturala(3x^3 + 2x^2 – 3x + 6) => 3*x^3 +2*x^2 -3*x^1 +6*x^0

# III. PROIECTARE

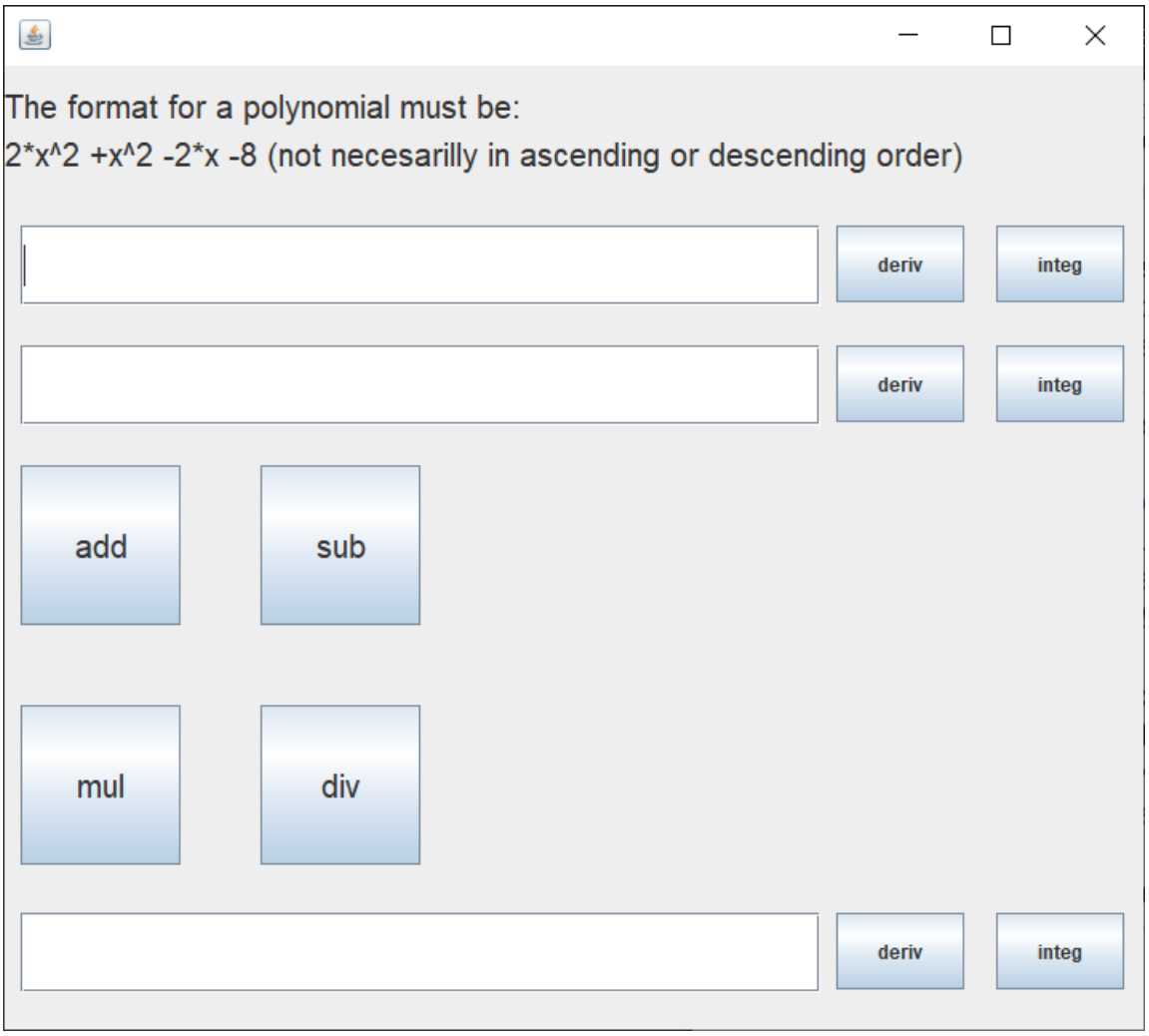
Pentru proiectarea aplicaţiei va fi nevoie de implementarea a doua clase imporntante, anume clasa Monom şi clasa Polinom. Aceste clase vor servi la stocare polinoamelor introduse de utilizator şi vor conţine şi de asemenea metode necesare pentru operaţii între monoame, respectiv polinoame.

Aici am realizat diagrama UML iniţială a programului. Clasa Polinom va conţine o listă de monoame şi metode pentru operaţii între polinoame. Clasa Monom conţine coeficient şi exponenet de tipul float, date necesare pentru descrierea completă a oricărui monom şi o serie de metode pentru lucrul cu monoame.

Clasa Constants va ţine o variabilă float utilozată pentru compararea între doua varoabile de tip float.



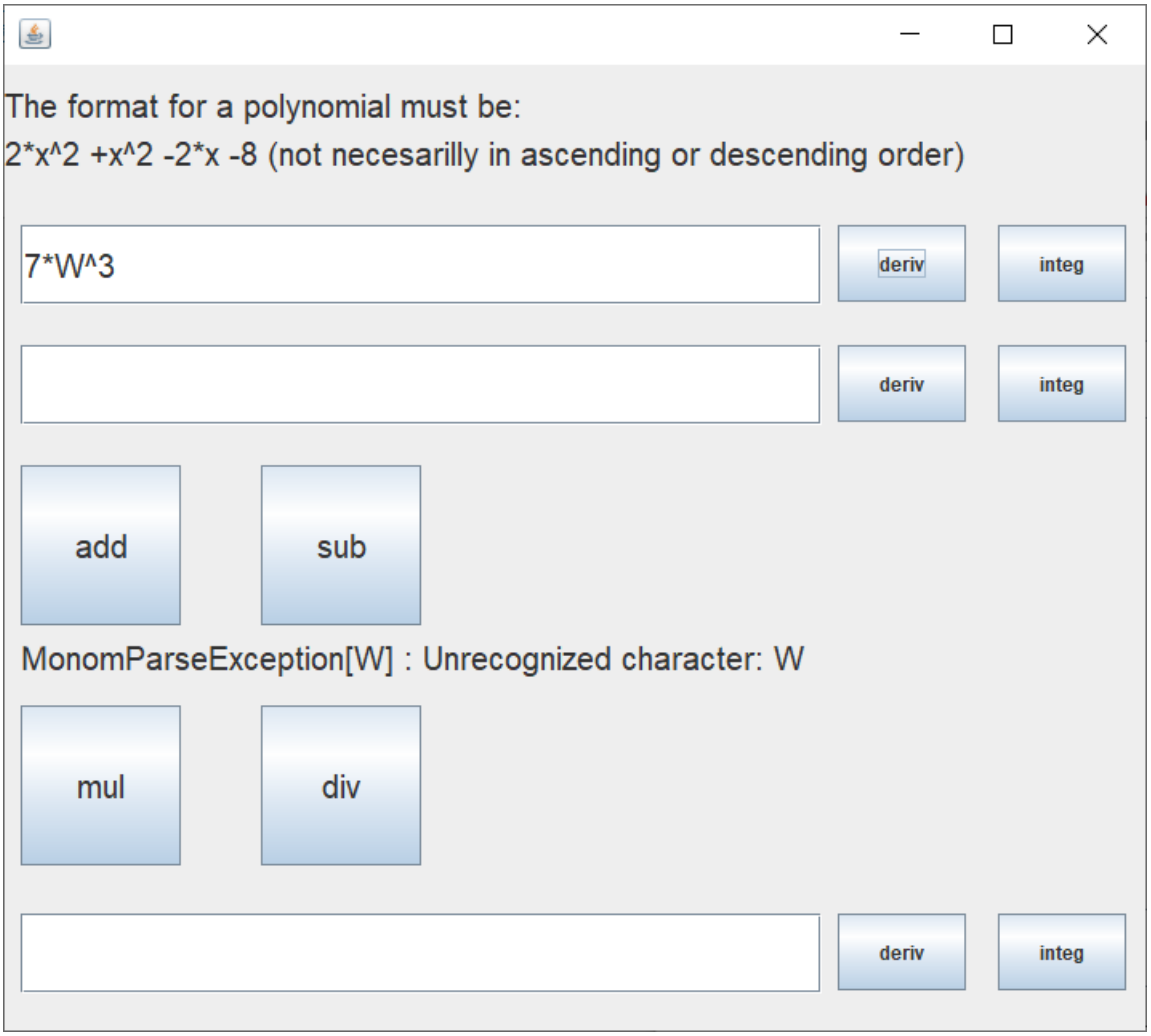
Pentru interfața grafică o să abordez un design simplu, punând accentul pe utilitate și funcționalitate.




Interfața cuprinde un exemplu de scriere a polinoamelor, doua TextField-uri unde ce vor introduce cele doua polinoame cu care dorim să lucră, și un al treilea TextField pentru rezultat. Fiecare TextField are în dreptul său  două butoane , *deriv* și *integ* care for efectua operațiile de derivare respectiv integrarea asupra polinomului din TextField. De asemenea, găsim cele patru butoane ce descriu operațiile între cele doua polinoame.

În cazul în care în scrierea polinomului apar caractere nepermise, acestea se vor semnaliza utilizarorului.

IV.




# IMPLEMENTARE

1. Clasa Monom

```
public class Monom implements Comparable<Monom>{

	private float exponent;
	private float coeficient;


	//addition
	public Monom add (Monom m)
	{
		Monom res = new Monom(0, 0);
		res.setCoeficient(coeficient + m.getCoeficient());
		res.setExponent(m.getExponent());
		return res;
	}
	//subtraction
	public Monom sub(Monom m)
	{
		Monom res = new Monom(0, 0);
```

```
	res.setCoeficient(coeficient - m.getCoeficient());
	res.setExponent(m.getExponent());
	return res;
}
//multiplication
public Monom mul(Monom m)
{
	Monom res = new Monom(0, 0);
	res.setCoeficient(coeficient * m.coeficient);
	res.setExponent(exponent + m.getExponent());
	return res;
}
//division
public Monom div(Monom m)
{
	Monom res = new Monom(0, 0);
	res.setCoeficient(coeficient / m.coeficient);
	res.setExponent(exponent - m.getExponent());
	return res;
}
//derivation
public Monom deriv()
{
	Monom res = new Monom(0, 0);
	if(Math.abs(exponent - 1) < Constants.eps)
	{
		res.setCoeficient(coeficient);
		return res;
	}
	else
	{
		res.setCoeficient(coeficient * exponent);
		res.setExponent(exponent - 1);
		return res;
	}

}
//integration
public Monom integ()
{
	Monom res = new Monom(1, 1);
	res.setCoeficient(coeficient / (exponent + 1));
	res.setExponent(exponent + 1);
	return res;
}

static public Monom sToM(String s) throws MonomParseException
{
	int flag = 0;
	Monom m = new Monom(1, 0);
	for(int i = 0; i < s.length(); i++)
	{
		if(s.charAt(i) == '*')
			continue;
		if(s.charAt(i) == '-')
			continue;
		if(s.charAt(i) == '+')
			continue;
```

```
                if(s.charAt(i) == '^')
                        continue;
                if(s.charAt(i) == 'x')
                        continue;
                if(s.charAt(i) == '.')
                        continue;
                if(s.charAt(i) == ' ')
                        continue;
                if((int)s.charAt(i) - (int)'0' >= 0 && (int)s.charAt(i) - (int)'0'< 10)
                        continue;
                flag = 1;
                throw(new MonomParseException(s.charAt(i)));
        }
        if(flag == 0)
        {
                // parsed string has format coeficient*x^exponent so res[0] = coef,
res[1] = x and res[2] = exponent
                String[] res = s.split(Pattern.quote("*"));
                if(res[0] == s)
                {
                        try
                        {
                                m.setCoeficient(Float.parseFloat(res[0]));
                        }
                        catch(Exception e)
                        {
                                if(res[0].charAt(0) == '-')
                                        m.setCoeficient(-1);
                                else
                                        m.setCoeficient(1);
                        }
                }
                else
                {
                        m.setCoeficient(Float.parseFloat(res[0]));
                }
                res = s.split(Pattern.quote("^"));
                if(res[0] == s)
                {
                        if(res[0].charAt(res[0].length() - 1) == 'x')
                        {
                                m.setExponent(1);
                        }
                        else
                        {
                                m.setExponent(0);
                        }

                }
                else
                {
                        m.setExponent(Float.parseFloat(res[1]));
                }
        }


        return m;
    }
```

```
	public String toString()
	{
		String s = "";
		if(coeficient >=0)
			s += "+";
		s += coeficient + "*x^" + exponent;
		return s;

	}


	public int compareTo(Monom m1) {

		if(this.getExponent() > m1.getExponent())
			return 1;
		if(this.getExponent() < m1.getExponent())
			return -1;
		return 0;
	}


}
```

Conţine doua variabile de tip float, coeficient şi exponent

Metoda public Monom sToM(String s) – primește un string și începe să îl parcurgă. Extrage coeficientul şi exponentul dar verifică dacă formatul este corect, în caz contrar aruncă o excepție.

Metoda public Monom add(Monom m) returnează un monom cu coeficientul, suma celorlalți coeficieți, analog și exponentul.

Metodele sub, mul, div, asemenătoare cu add.

2. Clasa Polinom

Conţine un ArrayList cu polinoame numit pol

Metoda public Polinom pToM(String s) – primește un String ce reprezintă un polinom. Metoda urmează să împartă string-ul în entități(monoame) pe care le trimite metodei sToM și introduce rezultatul apelurilor în în pol

Metode de add, sub, mul și div apelează după anumite reguli metodele cu aceleaşi nume din Meonm.

# V. REZULTATE

După testele efectuate în Junit am confirmat funcționalitatea corectă a funcțiilor de add, sub și mul. Funcția de div nu funționează.

```
package PT2019.demo.A1;

import java.util.ArrayList;

import junit.framework.*;
```

```
public class MainTest extends TestCase
{
	public Polinom p1 = new Polinom();
	public Polinom p2 = new Polinom();
	Monom m1 = new Monom(1, 2);
	Monom m2 = new Monom(2, 3);
	Monom m3 = new Monom(3, 3);
	Monom m4 = new Monom(1, 5);

	// setarea valorilor n1, n2
	public void setUp()
	{
		p1.add(m1);
		p1.add(m2);
		p2.add(m3);
		p2.add(m4);

		System.out.println("setUp:\n p1: " + p1 + "\n p2: " + p2);
	}
	// operatii cu cele doua valori
	public void testAdd()
	{
		Polinom p3 = new Polinom();
		p3 = p1;
		p3.add(p2);
		System.out.println(p3.toString());
		ArrayList<Monom> p = new ArrayList<Monom>();
		p.add(new Monom(1, 2));
		p.add(new Monom(5, 3));
		p.add(new Monom(1, 5));

		int flag = 1;
		for(int i = 0; i < p3.getSize(); i++)
		{
			if(p3.pol.get(i).getCoeficient() != p.get(i).getCoeficient() ||
p3.pol.get(i).getExponent() != p.get(i).getExponent())
				{
					flag = 0;

				}
		}
		assertTrue(flag == 1);
	}

	public void testSub()
	{
		Polinom p3 = new Polinom();
		p3 = p1;
		p3.sub(p2);
```

```
		ArrayList<Monom> p = new ArrayList<Monom>();
		p.add(new Monom(1, 2));
		p.add(new Monom(-1, 3));
		p.add(new Monom(-1, 5));
		int flag = 1;
		System.out.println(p3);
		for(int i = 0; i < p3.getSize(); i++)
		{
			if(p3.pol.get(i).getCoeficient() != p.get(i).getCoeficient() ||
p3.pol.get(i).getExponent() != p.get(i).getExponent())
				{
					flag = 0;

				}
		}
		assertTrue(flag == 1);
	}

	public void testMul()
	{
		Polinom p3 = new Polinom();
		p3 = p1;
		p3.mul(p2);
		System.out.println(p3);
		ArrayList<Monom> p = new ArrayList<Monom>();
		p.add(new Monom(3, 5));
		p.add(new Monom(6, 6));
		p.add(new Monom(1, 7));
		p.add(new Monom(2, 8));
		int flag = 1;
		for(int i = 0; i < p3.getSize(); i++)
		{
			if(p3.pol.get(i).getCoeficient() != p.get(i).getCoeficient() ||
p3.pol.get(i).getExponent() != p.get(i).getExponent())
				{
					flag = 0;

				}
		}
		assertTrue(flag == 1);
	}

	//metoda rulata dupa executarea testelor
	public void tearDown()
	{
		p1 = new Polinom();
		p2 = new Polinom();
		System.out.println("tearDown:\n p1: " + p1 + "\n p2: " + p2);
	}
```

}

În testare s-au luat doua polinoame și s-au adunat, scăzut și inmulțit. Rezultatul a fost comparat cu valoarea obținută efectiv pe foaie.

Mai jos avem rezultatele corecte obținute după rularea testelor în Junit pentru adunare, scădere și împărțire.

```
setUp:
 p1:  +x^2.0 +2.0x^3.0
 p2:  +3.0x^3.0 +x^5.0
 +x^2.0 +5.0x^3.0 +x^5.0
tearDown:
 p1:
 p2:
setUp:
 p1:  +x^2.0 +2.0x^3.0
 p2:  +3.0x^3.0 +x^5.0
 +3.0x^5.0 +6.0x^6.0 +x^7.0 +2.0x^8.0
tearDown:
 p1:
 p2:
setUp:
 p1:  +x^2.0 +2.0x^3.0
 p2:  +3.0x^3.0 +x^5.0
 +x^2.0 -1.0x^3.0 -1.0x^5.0
tearDown:
 p1:
 p2:
```

# VI. CONCLUZII

După finalizarea proiectul am realizat importanţa unei planificări solide, înainte de a începe efectiv scrierea codului, am învăţat importanţa utilizării variabilei epsilon la compararea numerelor float dar și complexitatea pe care o poate ascunde o cerinţă relativ simplă.

Pentru o dezvoltare ulteroioară, putem vorbi de implpementarea corectă a împărțirii, adaugarea unei sintaxe mai laxe la scrierea polinoamelor, adaugarea posibilității de utilizarea a polinoamelor de mai multe variabile dar și îmbunătățirea interfeței.

# VII. BIBLIOGRAFIE

https://www.tutorialspoint.com/What-are-custom-exceptions-in-Java

https://www.geeksforgeeks.org/searching-for-character-and-substring-in-a-string/
https://www.tutorialspoint.com/How-to-sort-an-ArrayList-in-Java-in-descending-order
http://www.anidescoala.ro/educatie/matematica/formule-matematice-algebra-liceu-generala/impartirea-polinoamelor/
https://www.w3schools.com/java/java_switch.asp
https://stackoverflow.com/questions/16458564/convert-character-to-ascii-numeric-value-in-java
http://zetcode.com/tutorials/javaswingtutorial/basicswingcomponentsII/
https://www.javamex.com/tutorials/swing/components.shtml
http://www.tutorialspoint.com/junit/junit_tutorial.pdf
http://www.junit.org
https://github.com/junit-team/junit/wiki/