Calculator de polinoame

Documentatie

Martin Maria-Denisa

Grupa 302210

Cuprins

1. Obiectivul temei.

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

3. Proiectare.

4. Implementare.

5. Rezultate.

6. Concluzii.

7.Bibliografie

1. Obiectivul temei:

Obiectivul principal al temei a fost proiectarea si implementarea unui calculator de polinoame.

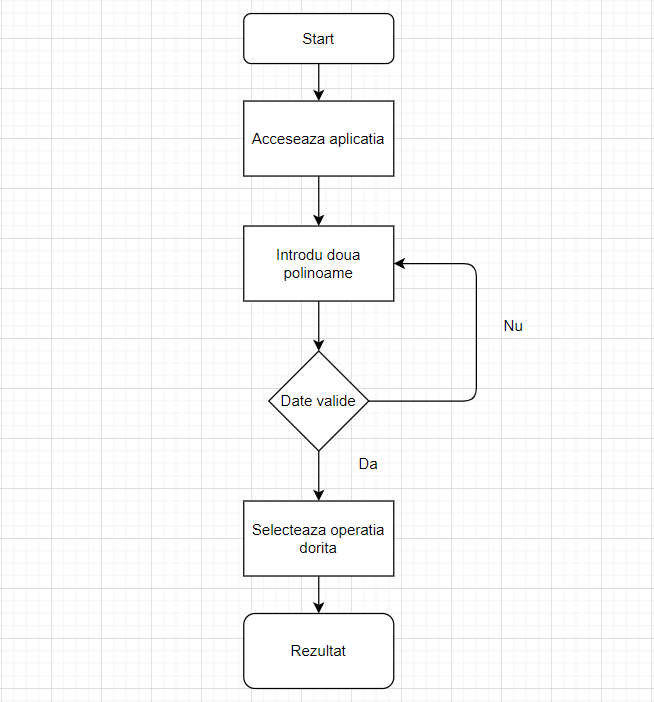
Polinoamele sunt construite din termeni numiți monoame, care sunt alcătuite dintr-o constantă (numită coeficient) înmulțită cu una sau mai multe variabile. Fiecare variabilă poate avea un exponent constant întreg pozitiv. Exponentul unei variabile dintr-un monom este egal cu gradul acelei variabile în acel monom. Coeficientul unui monom poate fi orice număr, inclusiv fracții, numere iraționale sau negative. Un polinom construit cu o singură variabilă se numește univariat. .

|  |  |
| --- | --- |
| Obiectiv secundar |  |
| Implementarea unei interfete grafice( clasa View, clasa Controller) | Este discutat la punctul 3 si 4 |
| Implementarea operatiilor pe monoame(clasa Monomial) | Este discutat la punctul 3 si 4 |
| Implementarea operatiilor pe polinoame: adunare, scadere, inmultire, impartire, derivare, integrare (clasa Polynimial) | Este discutat la punctul 3 si 4 |

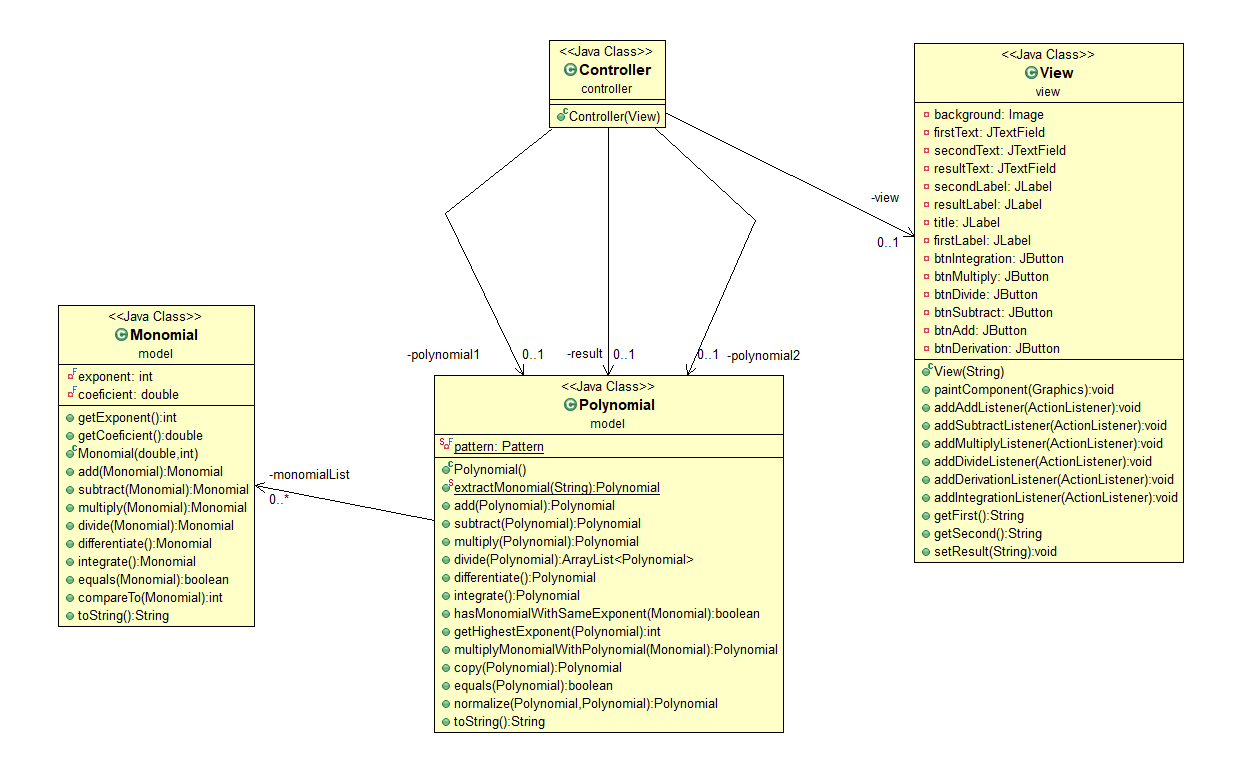
1. Analiza problemei, modelare, scenarii, cazuri de utilizare:

Programarea orientată ne permite să ne ocupam de problema de la un nivel superiror, fără a mai fi constrâși, într-o așa măsură, de caracteristicile tehnice. Această strategie mai poartă numele de bottom-up design. Este foarte avantajoasă deoarece pot fi găsite structuri cu o legătură directă în lumea reală( obiecte, acțiuni etc.). Punctul de pornire al proiectului este interfata cu utilizatorul, deoarece programul va putea fi accesat de un numar ridicat de persoane. Ea trebuie să permită comunicarea utilizatorului cu aplicația, sa fie usor de folosit si de inteles.În cazul Calculatorului de Polinoame, se cunoaște că aplicația trebuie să implementeze operațiile algebrice de adunare, scadere, impartire, inmultire, integrare si derivare. Acest lucru implică că trebuie să existe minim două polinoame. Pentru acestea am ales doua campuri de intrare si un camp de iesire unde se va returna rezultatul.

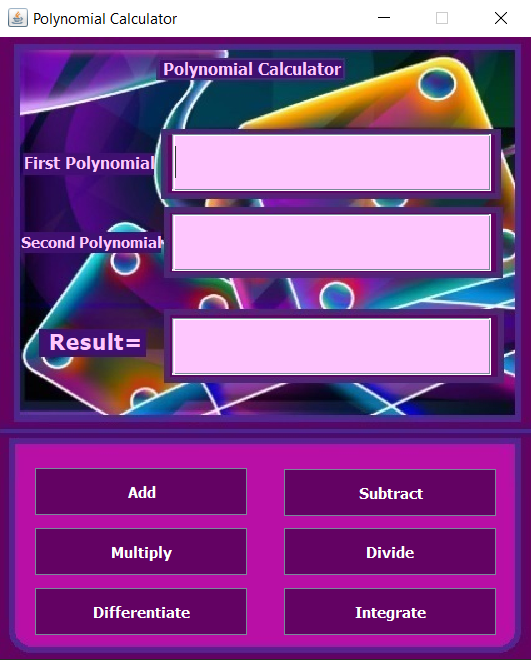
Din momentul stabilirii metodei de intrare accentul se va pune mai mult pe cazurile limită, care vor fi tratate mai special, precum: suma, diferenta, inmultirea și impărțirea între primul polinom și al doilea, derivarea si integrarea primului polinom, calcularea rezultatelor numerice pentru primul polinom și al doilea.Utilizarea normală a programului presupune introducerea primului si celui de al doilea polinom în formatul clasic, apoi se apasă butonul corespunzator operatiei dorite. Pentru operatiile de derivare si integrare este necesar doar ca primul polinom sa fie introdus. Totuși, cazul de mai sus este puțin probabil și adesea lucruri neașteptate pot să apară. Dintre acestea se remarcă :introducerea de text fără semnificație numerică, introducerea de necunoscute suplimentare(pe langă x), folosirea altor cuvinte cheie decât cele specificate, introducerea unui polinom dezordonat.



1. Proiectare (decizii de proiectare, diagrame UML, structure de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator).

Diagrama:

Interfata grafica:



Am decis sa proiectez clasele astfel incat sa fie cat mai intuitiv si mai lizibil codul. Calculatorul de Polinoame conține 4 clase de bază și o clasă main ce le instanțiază.

Clasa **Polynomial** – permite crearea polinoamelor ca liste de monoame, interpretarea polinoamelor de intrare, dar si operatiile dintre acestea.

Clasa **Monomial** – clasa fundamentală a proiectului ce definește structura de monom, entitate reprezentata de un coeficient si un exponent. Aici sunt prezentate si operatiile fundamentale asupra monoamelor: adunare, scadere, inmultire si impartire. Este implementata si o metoda toString(), care ajuta la afisarea cat mai frumoasa a monoamelor pe ecran.

Clasa **View** –cu ajutorul ei se creează interfața grafică folosită în tot proiectul. Tot aici se instanțiază și asculatătorii.

Clasa **Controller**- Aici sunt descrisi asculatorii. Ei au rolul de a surprinde eventualele acțiuni ale utilizatorilor prin intermediul evenimentelor. Tocmai de aceea, la nivelul butoanelor există câte un ascultător separat.

Pe lângă tipurile de date primitive existente în clasele specificate se folosesc și colecții. Colectia folosita se află în clasa Polynomial, ce reprezinta o listă de monoame. Există o relatie de compoziție între un polinom și elementele sale fundamentale, monoamele.

Am folosit modelul arhitectural MVC, in pachetul model sunt prezente clasele Monomial si Polynomial, in pachetul view este prezenta clasa View (care creeaza interfata grafica), iar in pachetul controller este prezenta clasa Controller care se ocupa de interactiuniile utilizatorului cu interfata prin implementarea ascultatorilor.

1. Implementare

Clasa View

Contine un JPanel pe care construieste toata interfata. S-a adoptat un layout de tip null, pentru a putea avea libertate in ceea ce priveste pozitia componentelor interfetei.

Au fost instantiate trei JtextField-uri , doua pentru datele de intrare si unul pentru afisarea rezultatului, dar si cele sase butoane pentru selectarea operatiei dorite.

firstText = new JTextField();  
firstText.setBackground(new Color(254, 199, 254));  
firstText.setFont(new Font("Tahoma", Font.*PLAIN*, 20));  
firstText.setBounds(139, 78, 255, 46);  
this.add(firstText);  
firstText.setColumns(10);

btnMultiply = new JButton("Multiply");  
btnMultiply.setBackground(new Color(99, 2, 99));  
btnMultiply.setFont(new Font("Tahoma", Font.*BOLD*, 12));  
btnMultiply.setBounds(29, 393, 170, 38);  
btnMultiply.setForeground(new Color(255, 255, 255));  
this.add(btnMultiply);

S-a folosti metoda paintComponent pentru a putea folosi o imagine ca si background.

public void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 if (background != null) {  
 g.drawImage(background, 0, 0, this);  
 }  
}

Clasa Controller

Contine ascultatorii pentru cele sase butoane. Ascultătorii sunt definit ca și clase imbricate cu anonimi.

this.view.addAddListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 polynomial1 = Polynomial.*extractMonomial*(view.getFirst());  
 polynomial2 = Polynomial.*extractMonomial*(view.getSecond());  
 result = polynomial1.add(polynomial2);  
 view.setResult(result.toString());  
 }  
});

Clasa Monomial

Clasa care conține operațiile elementare de adunare, scădere, înmulțire, împărțire, derivare, integrare asupra monoamelor. Un monom este reprezentat de un coeficient de tip double si de un exponent intreg.

public class Monomial implements Comparable<Monomial>{  
 private final int exponent;  
 private final double coeficient;

public Monomial add(Monomial monom1) {  
 return new Monomial(this.coeficient + monom1.coeficient, this.exponent);  
}  
  
public Monomial subtract(Monomial monom1) {  
 return new Monomial(this.coeficient - monom1.coeficient, monom1.exponent);  
}  
  
public Monomial multiply(Monomial monom) {  
 return new Monomial(this.coeficient \* monom.coeficient, this.exponent + monom.exponent);  
}  
  
public Monomial divide(Monomial monom) {  
 return new Monomial(this.coeficient / monom.coeficient, this.exponent - monom.exponent);  
}  
  
public Monomial differentiate() {  
 return new Monomial(this.getCoeficient() \* this.getExponent(),this.getExponent() - 1 );  
}  
  
public Monomial integrate() {  
 return new Monomial(this.getCoeficient() / (this.getExponent() + 1), this.getExponent() + 1);  
}

Contine metoda equals pentru testarea egalitatii a doua monoame. Doua monoame sunt egale daca au acelasi coeficient si acelasi exponent.

public boolean equals(Monomial b) {  
 return (this.coeficient == b.getCoeficient()) && (this.exponent == b.getExponent());  
}

Implemeteza metoda compareTo din Comparable, care ajuta la sortarea monoamelor dupa exponent.

public int compareTo(Monomial x) {  
 Integer exponent1= this.exponent;  
 Integer exponent2= x.getExponent();  
 return exponent2.compareTo(exponent1);  
}

Contine si metoda toString pentru afisarea monoamelor pe ecran, unde au fost luate in considerare toate cazurile: coeficient pozitiv, coeficient negativ, coeficienti intregi sau reali, eponent 0, exponent diferit de 0.

public String toString() {  
 if (this.coeficient == (int) this.coeficient) {  
 if (this.coeficient >= 0) {  
 if (this.exponent != 0) {  
 return "+" + (int) this.coeficient + "x^" + this.exponent;  
 } else {  
 return "+" + (int) this.coeficient;  
 }  
 } else {  
 if (this.exponent != 0) {  
 return (int) this.coeficient + "x^" + this.exponent;  
 } else {  
 return ""+ (int) this.coeficient;  
 }  
 }  
 } else {  
 if (this.coeficient >= 0) {  
 if (this.exponent != 0) {  
 return "+" + this.coeficient + "x^" + this.exponent;  
 } else {  
 return "+" + this.coeficient;  
 }  
 } else {  
 if (this.exponent != 0) {  
 return + this.coeficient + "x^" + this.exponent;  
 } else {  
 return "" + this.coeficient;  
 }  
  
 }  
 }  
}

Clasa Polynomial

Este reprezentata de un ArrayList de monoame. Contine metoda extractMonomial care extrage coeficentii si exponentii monoamelor care alcatuiesc polinomul de intrare pentru a creea lista de monoame care implicit reprezinta polinomul nostru.

public static Polynomial extractMonomial(String input) {  
 Matcher matcher = *pattern*.matcher(input);  
 double coeficient;  
 int exponent = 0;  
 Polynomial polynomial = new Polynomial();  
 while (matcher.find()) {  
 if (matcher.group(3) != null && matcher.group(2) != null) {  
 coeficient = Double.*parseDouble*(matcher.group(2));  
 exponent = (matcher.group(5) != null ? Integer.*parseInt*(matcher.group(5)) : 1);  
  
 } else {  
 coeficient = Integer.*parseInt*(matcher.group());  
 }  
 Monomial monomial = new Monomial(coeficient, exponent);  
 polynomial.monomialList.add(monomial);  
 exponent = 0;  
 }  
 Collections.*sort*(polynomial.monomialList);  
 return polynomial;  
}

Implementeaza o metoda pentru a verifica daca o putere se regaseste deja in lista noastra de monoame. Returneaza true daca exponentul este in lista si false in caz contrar.

public boolean hasMonomialWithSameExponent(Monomial monomial) {  
 for (Monomial monom : this.monomialList) {  
 if (monomial.getExponent() == monom.getExponent()) {  
 return true;  
 }  
 }  
 return false;  
}

Metoda getHighestExponent returneaza exponentului maxim al polinomului.

public int getHighestExponent(Polynomial polynomial) {  
 int max = -9999;  
 for (Monomial monom : polynomial.monomialList) {  
 if (max < monom.getExponent()) {  
  
 max = monom.getExponent();  
 }  
 }  
 return max;  
}

Metoda multiplyMonomialWithPolynomial care inmulteste un monom cu un polllinom.

public Polynomial multiplyMonomialWithPolynomial(Monomial monomial) {  
 Polynomial result = new Polynomial();  
 for (Monomial monom : this.monomialList) {  
 result.monomialList.add(monom.multiply(monomial));  
 }  
 return result;  
}

Metoda copy returneaza copia unui polinom.

public Polynomial copy(Polynomial polynomial) {  
 Polynomial result = new Polynomial();  
 result.monomialList.addAll(polynomial.monomialList);  
 return result;  
}

Metoda add returneaza rezultatul adunarii. Parcurgem cele doua polinoame, daca doua monoame au acelasi exponent le adunam si le scoatem din polinom, la final adaugam la rezultat ce a mai ramas din cele doua polinoame si il sortam descrescator dupa exponent.

public Polynomial add(Polynomial polynomial) {  
 Polynomial result = new Polynomial();  
 Polynomial polynomial1 = copy(this);  
 Polynomial polynomial2 = copy(polynomial);  
 int i = 0;  
 int j = 0;  
 while (i < this.monomialList.size() && j < polynomial.monomialList.size()) {  
 Monomial monomial1 = this.monomialList.get(i);  
 Monomial monomial2 = polynomial.monomialList.get(j);  
 if (monomial1.getExponent() == monomial2.getExponent()) {  
 polynomial1.monomialList.remove(monomial1);  
 polynomial2.monomialList.remove(monomial2);  
 result.monomialList.add(monomial1.add(monomial2));  
 i++;  
 j++;  
 } else {  
 if (monomial1.getExponent() > monomial2.getExponent()) {  
 i++;  
 } else {  
 j++;  
 }  
 }  
 }  
 result.monomialList.addAll(polynomial1.monomialList);  
 result.monomialList.addAll(polynomial2.monomialList);  
 Collections.*sort*(result.monomialList);  
 return result;  
}

Metoda subtract returneaza diferenta dintre doua polinoame. Parcurgem cele doua polinoame, daca doua monoame au acelasi exponent le scadem si le scoatem din polinom, la final adaugam la rezultat ce a mai ramas din cele doua polinoame si il sortam descrescator dupa exponent.

public Polynomial subtract(Polynomial polynomial) {  
 Polynomial result = new Polynomial();  
 Polynomial polynomial1 = copy(this);  
 Polynomial polynomial2 = copy(polynomial);  
 int i = 0;  
 int j = 0;  
 while (i < this.monomialList.size() && j < polynomial.monomialList.size()) {  
 Monomial monomial1 = this.monomialList.get(i);  
 Monomial monomial2 = polynomial.monomialList.get(j);  
 if (monomial1.getExponent() == monomial2.getExponent()) {  
 polynomial1.monomialList.remove(monomial1);  
 polynomial2.monomialList.remove(monomial2);  
 if (monomial1.getCoeficient() != monomial2.getCoeficient())  
 result.monomialList.add(monomial1.subtract(monomial2));  
 i++;  
 j++;  
 } else {  
 if (monomial1.getExponent() > monomial2.getExponent()) {  
 i++;  
 } else {  
 j++;  
 }  
 }  
 }  
 result.monomialList.addAll(polynomial1.monomialList);  
 result.monomialList.addAll(polynomial2.monomialList);  
 Collections.*sort*(result.monomialList);  
 return result;  
}

Metoda multiply returneaza rezultatul inmultirii a doua polinoame. Fiecare element din primul polinom este inmultit cu fiecare element din cel de al doilea polinom. Apoi se foloseste o metoda de normalizare prin care se cauta in rezultat daca sunt monoame cu acelasi exponent, daca da, acestea sunt adunate.

public Polynomial multiply(Polynomial polynomial) {  
 Polynomial normalizedPolynomial = new Polynomial();  
 Polynomial intermediar = new Polynomial();  
 for (Monomial monomial1 : this.monomialList) {  
 for (Monomial monomial2 : polynomial.monomialList) {  
 intermediar.monomialList.add(monomial1.multiply(monomial2));  
 }  
 }  
 normalizedPolynomial = normalize(intermediar, normalizedPolynomial);  
 return normalizedPolynomial;  
}

Metoda normalize:

public Polynomial normalize(Polynomial intermediar, Polynomial normalizedPolynomial) {  
 for (int i = 0; i < intermediar.monomialList.size(); i++) {  
 if (!normalizedPolynomial.hasMonomialWithSameExponent(intermediar.monomialList.get(i))) {  
 Monomial rezultatSumaInter = intermediar.monomialList.get(i);  
 for (int j = i + 1; j < intermediar.monomialList.size() - 1; j++) {  
 if (intermediar.monomialList.get(i).getExponent() == intermediar.monomialList.get(j).getExponent()) {  
 rezultatSumaInter = rezultatSumaInter.add(intermediar.monomialList.get(j));  
 }  
 }  
 normalizedPolynomial.monomialList.add(rezultatSumaInter);  
 }  
 }  
 return normalizedPolynomial;  
}

Metoda divide returneza rezultatul impartirii a doua polinoame sub forma unei liste de polinoame, in care primul rezultat reprezinta catul impartirii, iar al doilea reprezinta restul.

public ArrayList<Polynomial> divide(Polynomial polynomial) {  
 ArrayList<Polynomial> result = new ArrayList<>();  
 Polynomial quotient = new Polynomial();  
 Polynomial remainder = this;  
 while ((!remainder.monomialList.isEmpty()) && (getHighestExponent(remainder) >= getHighestExponent(polynomial))) {  
 Monomial monomial=remainder.monomialList.get(0).divide(polynomial.monomialList.get(0));  
 quotient.monomialList.add(monomial);  
 remainder = remainder.subtract(polynomial.multiplyMonomialWithPolynomial(monomial));  
 }  
 result.add(quotient);  
 result.add(remainder);  
 return result;  
}

Metoda differentiate returneaza primul polinom derivat.

public Polynomial differentiate() {  
 Polynomial result = new Polynomial();  
 for (Monomial monom : this.monomialList) {  
 if (monom.getExponent() != 0) {  
 result.monomialList.add(monom.differentiate());  
 }  
 }  
 return result;  
}

Metoda integrate returneaza primul polinom integrat.

public Polynomial integrate() {  
 Polynomial result = new Polynomial();  
 for (Monomial monom : this.monomialList) {  
 result.monomialList.add(monom.integrate());  
 }  
 return result;  
}

Clasa MainClass

Contine metoda Main, prin care sunt instantiate celelate clase.

public class MainClass {  
 public static void main(String[] args) throws IOException {  
 JFrame obj = new JFrame();  
 View view = new View("background1.jpg");  
 obj.getContentPane().add(view);  
 obj.setBounds(100, 100, 440, 535);  
 Controller controller = new Controller(view);  
 obj.setTitle("Polynomial Calculator");  
 obj.setResizable(false);  
 obj.setVisible(true);  
 obj.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
 }  
}

5.Rezultate

Programul ruleaza corespunzător chiar daca polinoamele nu sunt introduse in ordinea corecta a exponentilor. Rezultatele, cât și valorile se află în grila celor așteptate. Pentru testare s-a folosit Junit. Cazuri testate cu junit:

public class PolynomialTest {  
 Polynomial polymonial1 =new Polynomial();  
 Polynomial polynomial2 =new Polynomial();  
 Polynomial result= new Polynomial();  
  
 @Test  
 public void addAddTest(){  
 polymonial1 = Polynomial.*extractMonomial*("2x^2+1");  
 polynomial2 = Polynomial.*extractMonomial*("2x^2");  
 result= Polynomial.*extractMonomial*("4x^2+1");  
 *assertTrue*(polymonial1.add(polynomial2).equals(result));  
 }  
 @Test  
 public void addSubtractTest(){  
 polymonial1 = Polynomial.*extractMonomial*("2x^2+1");  
 polynomial2 = Polynomial.*extractMonomial*("1x^2");  
 result= Polynomial.*extractMonomial*("1x^2+1");  
 *assertTrue*(polymonial1.subtract(polynomial2).equals(result));  
  
 }  
 @Test  
 public void addMultiplyTest(){  
 polymonial1 = Polynomial.*extractMonomial*("2x^2+1");  
 polynomial2 = Polynomial.*extractMonomial*("1x^2");  
 result= Polynomial.*extractMonomial*("2x^4+1x^2");  
 *assertTrue*(polymonial1.multiply(polynomial2).equals(result));  
  
 }  
 @Test  
  
 public void addDivideTest(){  
 ArrayList<Polynomial> rez=new ArrayList<>();  
 polymonial1 =Polynomial.*extractMonomial*("2x^4");  
 polynomial2 = Polynomial.*extractMonomial*("1x^2");  
 rez.add(Polynomial.*extractMonomial*("2x^2"));  
 rez.add(new Polynomial());  
 *assertTrue*(polymonial1.divide(polynomial2).get(0).equals(rez.get(0))&& polymonial1.divide(polynomial2).get(1).equals(rez.get(1)));  
  
 }  
  
 @Test  
 public void addDerivativeTest(){  
  
 polymonial1 =Polynomial.*extractMonomial*("1x^3−2x^2+6x^1−5");  
 result=Polynomial.*extractMonomial*("3x^2−4x^1+6");  
 *assertTrue*(polymonial1.differentiate().equals(result));  
  
 }  
 @Test  
 public void addIntegrateTest(){  
  
 polymonial1 =Polynomial.*extractMonomial*("6x^2+1");  
 result=Polynomial.*extractMonomial*("2x^3+1x^1");  
 *assertTrue*(polymonial1.integrate().equals(result));  
  
 }  
  
}  
}

Mai multe situatii au fost testate practic, chiar prin introducerea lor de la tastatura.

6.Concluzii

  Din această primă temă, am invatat care este rolul pe care separarea obiectivelor îl are chiar la începutul problemei, cum se separa coeficientii si exponentii unui polinom dat sub forma unui String. Aceasta tema a intarit si aprofundat cunostiintele mele despre limbajul Java.

Ca și dezvoltări ulteriore, ideea de a afla radacinile unui polinom pare foarte folositoare, dar si afisarea unor grafice reprezentative.

7.Bibliografie

<https://ro.wikipedia.org/wiki/Polinom>