Documentatie Tema 1

-Calculator polinoame-

Nume: Hilbert Dennis

Seria: B, Grupa: 30226 An 2, CTI

Cuprins

Obiectivul temei . . . . . . . . . . . . . . . . . . . . . . . . . 3

Proiectare . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

-Descrierea generala a proiectului . . . . . . . 3

-Diagrama UML . . . . . . . . . . . . . . . . . . . . . . 4

-Descrierea claselor . . . . . . . . . . . . . . . . . . . 4

Implementare . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

Rezultate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

Concluzii . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

Bibliografie . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

1. Obiectivul temei

Obiectivul temei consta in implementarea unei aplicatii care realizeaza operatii pe polinoame.

Acesta este prevazut cu o interfata grafica ce permite introducerea polinoamelor de catre utilizator si alegerea operatiilor, pentru ca mai apoi sa furnizeze un rezultat corespunzator.

Obiectivele secundare sunt reprezentate de crearea si structurarea intr-un mod avantajos a pachetelor, claselor si functiilor care le compun. Acestea ar fi, in ordine, urmatoarele:

-crearea functiilor care transforma datele de tip String in obiecte de tip Monom, respectiv Polinom

-prelucrarea datelor prin intermediul functiilor de adunare, scadere, inmultire, impartire, integrare si derivare

-crearea unei interfete grafice pentru utilizator

-stabilirea legaturilor intre interfata si functii

-testarea aplicatiei cu ajutorul Juni

Acestea urmeaza sa fie detaliate in capitolul 3 care descrie in detaliu proiectarea aplicatiei.

2.Proiectare

2.1Descrierea generala a proiectului:

Din punct de vedere structural, proiectul contine doua pachete si anume Core si MVC.

In pachetul Core se gasesc clasele Monom si Polinom, iar in pachetul MVC se gasesc clasele Model, View , Controller si Main. Pe langa acestea, proiectul contine un folder de test in care sunt verificate functiile aplicabile pe polinoame.

Din punct de vedere al functionalitatii, metodele de prelucrare ce descriu operatiile pe polinoame se afla toate in clasa Model. Aceasta are 3 atribute de tip Polinom: rezultat, rezultatDiv si rest, ultimele doua fiind folosite doar in cazul operatiei de impartire, intrucat rezultatul acestei operatii are o forma diferita fata de celelalte. Clasa View descrie forma interfetei grafice, iar clasa Controller realizeaza legatura intre Model si View. Acestea sunt toate reunite in Main cu scopul de a definitiva aplicatia.

Clasele Monom si Polinom contin in mare parte gettere si settere pentru accesul la atribute dar si unele functii care ajuta la implementare(de exemplu, degree, isGreaterThan, printList, etc.).

2.2Diagrama UML a proiectului:

O imagine care conține hartă, text

Descriere generată automat

2.3 Descrierea claselor:

**Pachetul Core:**

-Clasa Monom implements Comparable<Monom>

* Atribute:
  + private double coef
  + private int putere
* Constructori:
  + public Monom()
  + public Monom(double, int) : se foloseste pentru a crea un nou monom cu coeficient si putere specifice
  + public Monom(String) : se foloseste pentru a trece de la un String la format de monom
* Metode:

public double getCoef()  
  
public void setCoef(double coef)   
public int getPutere()  
  
public void setPutere(int putere)

public int compareTo(Monom m)

-Clasa Polinom

* Atribute:
  + - private ArrayList<Monom> mon\_list;
* Constructori:
  + - Polinom()
    - Polinom(String) : se foloseste pentru a imparti un String in mai multe String-uri ce au forma de monom, acestea fiind transmise mai departe constructorului din clasa Monom
    - Polinom(Polinom) : se foloseste pentru a crea un nou polinom identic cu cel primit ca parametru
* Metode:

public void setMon\_list(ArrayList<Monom>)  
  
 public void polAdjust(Polinom)  
  
 public void printList()  
  
 public int degree()  
  
 public int isGreaterThan()

**Pachetul MVC:**

-Clasa Model

* Atribute:
  + - private Polinom rezultat
    - private Polinom rezultatDiv
    - private Polinom rest
* Metode

public void polynomialAddition(Polinom p, Polinom q)

public void polynomialSubstraction(Polinom p, Polinom q)

public void polynomialMultiplication(Polinom p, Polinom q)

public void polynomialDivision(Polinom p, Polinom q)

public void polynomialIntegration(Polinom p)

public void polynomialDerivative(Polinom p)   
  
  
 public String getRezultat()

public String getRezultatDiv()

-Clasa View extends JFrame

* Atribute:

private JTextField firstPolynomial = new JTextField(20);  
private JTextField secondPolynomial = new JTextField(20);  
private JTextField result = new JTextField(40);  
private JLabel firstPText = new JLabel("Primul polinom: ");  
private JLabel secondPText = new JLabel(" Al doilea polinom: ");  
private JLabel opText = new JLabel("Operatii: ");  
private JLabel resText = new JLabel("Rezultat: ");  
private JButton addition = new JButton("+");  
private JButton substraction = new JButton("-");  
private JButton multiplication = new JButton("\*");  
private JButton division = new JButton("/");  
private JButton integration = new JButton("\u222B");  
private JButton derivative = new JButton("'");

* Constructori:

public View() {  
 this.setSize(700, 180);  
 this.setTitle("Polynomial Calculator");  
  
 JPanel polynomialPanel = new JPanel();  
 polynomialPanel.add(firstPText);  
 polynomialPanel.add(firstPolynomial);  
 polynomialPanel.add(secondPText);  
 polynomialPanel.add(secondPolynomial);  
 this.add(polynomialPanel, BorderLayout.*PAGE\_START*);  
  
 JPanel buttonPanel = new JPanel();  
 buttonPanel.add(opText);  
 buttonPanel.add(addition);  
 buttonPanel.add(substraction);  
 buttonPanel.add(multiplication);  
 buttonPanel.add(division);  
 buttonPanel.add(integration);  
 buttonPanel.add(derivative);  
 this.add(buttonPanel, BorderLayout.*CENTER*);  
  
 JPanel resultPanel = new JPanel();  
 resultPanel.add(resText);  
 resultPanel.add(result);  
 this.add(resultPanel, BorderLayout.*PAGE\_END*);  
  
 this.setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);  
  
 this.setVisible(true);  
}

* Metode:

String getUserFirstInput()   
 String getUserSecondInput()   
  
 void setResult(String newTotal)   
 void showError(String errMessage)   
 void addAdditionListener(ActionListener a)

void addSubstractionListener(ActionListener b)  
 void addMultiplicationListener(ActionListener c)  
 void addDivisionListener(ActionListener d)   
 void addIntegrationListener(ActionListener e)   
 void addDerivativeListener(ActionListener f)

-Clasa Controller

Atribute:

private Model m\_model;  
private View m\_view;

Constructori:

public Controller(Model model, View view) {  
 this.m\_model = model;  
 this.m\_view = view;  
  
 m\_view.addAdditionListener(new AdditionListener());  
 m\_view.addSubstractionListener(new SubstractionListener());  
 m\_view.addMultiplicationListener(new MultiplicationListener());  
 m\_view.addDivisionListener(new DivisionListener());  
 m\_view.addIntegrationListener(new IntegrationListener());  
 m\_view.addDerivativeListener(new DerivativeListener());  
}

Subclase si metode:

class AdditionListener implements ActionListener {

public void actionPerformed(ActionEvent e)

}  
class SubstractionListener implements ActionListener {

public void actionPerformed(ActionEvent e)

}  
class MultiplicationListener implements ActionListener {

public void actionPerformed(ActionEvent e)

}  
class DivisionListener implements ActionListener {  
 public void actionPerformed(ActionEvent e)

}  
class IntegrationListener implements ActionListener {  
 public void actionPerformed(ActionEvent e)

}  
class DerivativeListener implements ActionListener {  
 public void actionPerformed(ActionEvent e)

}

-Clasa Main

public static void main(String[] args) {  
 View v\_view = new View();  
 Model m\_model = new Model();  
 Controller Calculator = new Controller(m\_model, v\_view);  
}

3.Implementare

-Clasa Monom

Cele mai importante componente ale clasei Monom sunt constructorii. Acestia ajuta in a crea elemente cu proprietati de monom pe baza parametrilor primiti.

public Monom(String s) {  
 int nr = 0;  
 for (String val: s.split( "x\\^")) {  
 //System.out.println("valoare coef + putere: " + val);  
 if(nr % 2 == 0)  
 this.coef = Double.*parseDouble*(val);  
 else if(nr % 2 == 1)  
 this.putere = Integer.*parseInt*(val);  
 nr++;  
 }  
}

-In cazul in care parametrul primit este un String, prelucrarea se face cu Regex pentru a delimita coeficientii is puterile de „x” si de simbolul de putere „^”. Fiecare monom din cadrul polinomului trebuie sa aiba specificat un coeficient de tip double urmat de „x^” iar mai apoi de puterea reprezentata printr-un numar intreg.

In rest, clasa contine gettere si settere pentru accesul la coeficienti si puteri dar si o metoda compareTo care ajuta la compararea monoamelor. Aceasta este utila pentru algoritmii care au nevoie de sortari.

-Clasa Polinom

Clasa polinom detine ca singur atribut un ArrayList de tip Monom care reprezinta efectiv lista elementelor polinomului.

Se asteapta ca String-ul introdus de utilizator sa reprezinte un polinom in care sunt specificati toti coeficientii(in afara de cei egali cu 0) si toate puterile.

De exemplu, un String de genul „2x^3+1x^0” este coniderat valid, pe cand unul ca”2x^2+x^0” nu este valid deoarece lipseste coeficientul lui x^0.

In cazul in care primul coeficient al polinomului lipseste, va fi aruncata o exceptie.

public Polinom(String s) {  
 mon\_list = new ArrayList<Monom>();  
 Pattern pattern = Pattern.*compile*("([-]?[^+-]+)");  
 Matcher matcher = pattern.matcher(s);  
 int x = 0;  
 while (matcher.find()) {  
 x = x + 1;  
 Monom m = new Monom(matcher.group());  
 if (!mon\_list.isEmpty() && mon\_list.get(mon\_list.size() - 1).getPutere() - 1 != m.getPutere()) {  
 int nr = mon\_list.get(mon\_list.size() - 1).getPutere() - m.getPutere() - 1;  
 while (nr != 0) {  
 mon\_list.add(new Monom(0, mon\_list.get(mon\_list.size() - 1).getPutere() - 1));  
 nr--;  
 }  
 }  
 mon\_list.add(m);  
 }  
 while (mon\_list.get(mon\_list.size() - 1).getPutere() != 0)  
 mon\_list.add(new Monom(0, mon\_list.get(mon\_list.size() - 1).getPutere() - 1));  
  
}

-In momentul in care Constructorul clasei primeste ca parametru un String, acesta este impartit utilizant Regex in mai multe String-uri corespunzatoare unor monoame, acestea fiind transmise mai departe constructorului din clasa Monom, care stabileste mai departe coeficientii si puterile.

Dupa ce textul este procesat, monomul este adaugat in lista de monoame a polinomului, iar la final, toate elementele ce „lipsesc” din poliom(cele cu coeficienti 0) vor fi adaugate in lista de asemenea pentru a ajuta in momentul in care se fac operatii.

Printre metodele importante se numara si:

* degree: returneaza gradul polinomului
* isGreaterThan:determina daca un polinom este mai mare decat altul
* polAdjust: utilizat pentru a elimina bug-urile ce apareau la impartire

-Clasa Model

Aceasta clasa contine toate operatiile ce se realizeaza pe polinoame. Are 3 atribute: unul universal (rezultat – reprezinta in functie de alegerea utilizatorului rezultatul din urma executarii oricarei metode in afara de impartire) si doua particulare (rezultatDiv si rest-reprezinta rezultatul din urma impartirii a doua polinoame).

Asadar, clasa model contine urmatoarele metode:

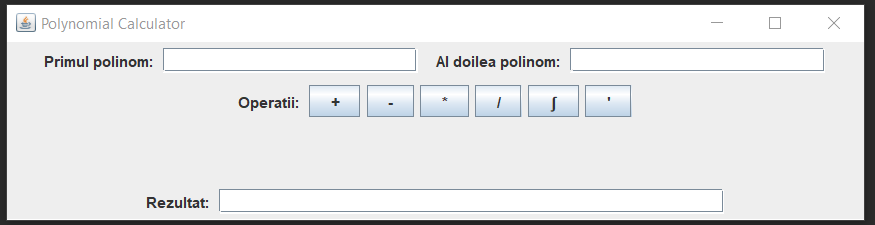
* public void polynomialAddition (Polinom p, Polinom q) : calculeaza suma a doua polinoame si salveaza rezultatul in atributul rezultat (rezultat = p + q)
* public void polynomialSubstraction (Polinom p, Polinom q) : calculeaza diferenta a doua polinoame si salveaza rezultatul in atributul rezultat (rezultat = p - q)
* public void polynomialMultiplication (Polinom p, Polinom q) : calculeaza produsul a doua polinoame si salveaza rezultatul in atributul rezultat (rezultat = p \* q)
* public void polynomialDivision(Polinom p, Polinom q) : realizeaza impartirea a doua polinoame si salveaza rezultatul in atributele rezultatDiv si rest (rezultatDiv + rest = p / q)
* public void polynomialIntegration(Polinom p) : calculeaza integrala polinomului p – primul polinom introdus de utilizator
* public void polynomialDerivative(Polinom p) : calculeaza derivata polinomului p – primul polinom introdus de utilizator

Metodele prin care se furnizeaza rezultatele sub forma de string sunt:

public String getRezultat() {  
 String s = "";  
 Collections.*sort*(rezultat.getMon\_list());  
 rezultat.getMon\_list().removeIf(Monom -> Monom.getCoef() == 0);  
 for (Monom m : rezultat.getMon\_list()) {  
 s += m.getCoef() + "x^" + m.getPutere();  
 if (rezultat.getMon\_list().indexOf(m) != rezultat.getMon\_list().size() - 1)  
 if (rezultat.getMon\_list().get(rezultat.getMon\_list().indexOf(m) + 1).getCoef() > 0)  
 s += "+";  
 }  
 if (s == "")  
 s += "0";  
 return s;  
}  
  
public String getRezultatDiv() {  
 String s = "";  
 Collections.*sort*(rezultatDiv.getMon\_list());  
 rezultatDiv.getMon\_list().removeIf(Monom -> Monom.getCoef() == 0);  
 for (Monom m : rezultatDiv.getMon\_list()) {  
  
 s += m.getCoef() + "x^" + m.getPutere();  
 if (rezultatDiv.getMon\_list().indexOf(m) != rezultatDiv.getMon\_list().size() - 1)  
 if (rezultatDiv.getMon\_list().get(rezultatDiv.getMon\_list().indexOf(m) + 1).getCoef() > 0)  
 s += "+";  
 }  
 if (s == "")  
 s += "-";  
 s += ", rest: ";  
 Collections.*sort*(rest.getMon\_list());  
 rest.getMon\_list().removeIf(Monom -> Monom.getCoef() == 0);  
 for (Monom m : rest.getMon\_list()) {  
 s += m.getCoef() + "x^" + m.getPutere();  
 if (rest.getMon\_list().indexOf(m) != rest.getMon\_list().size() - 1)  
 if (rest.getMon\_list().get(rest.getMon\_list().indexOf(m) + 1).getCoef() > 0)  
 s += "+";  
 }  
 if (rest.degree() < 0)  
 s += "0" return s;  
}

-Clasa View

Clasa View ca atribute toate elementele necesare realizarii interfetei grafice, un constructor in care se creeaza JPanel-uri carora li se ataseaza elementele necesare utilizarii + metode care ajuta la preluarea de stringuri si cate o metoda de tip Listener pentru fiecare buton respectiv functie in parte.



Interfata contine 3 JPanel-uri organizate cu BorderLayout pe pozitiile PAGE\_START, CENTER SI PAGE\_END

-Clasa Controller

Clasa Controller are ca atribute :

private Model m\_model;  
private View m\_view;

Constructorul primeste un parametru de tip Model si unul de tip View si se adauga cate un nou Listener pentru fiecare functie din clasa atributului m\_model, realizand astfel legatura intre interfata si functionalitatea acesteia.

Urmatoarele componente sunt alte 6 clase(pentru fiecare operatie) realizate dupa modelul AdditionListener de mai jos care primest un firstInput si secondInput(daca este cazul) si furnizeaza inapoi rezultatul cu ajutorul metodei setResult.

class AdditionListener implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 String firstInput = "";  
 String secondInput = "";  
 try {  
 firstInput = m\_view.getUserFirstInput();  
 secondInput = m\_view.getUserSecondInput();  
 m\_model.polynomialAddition(new Polinom(firstInput), new Polinom(secondInput));  
 m\_view.setResult(m\_model.getRezultat());  
  
 } catch (NumberFormatException nfex) {  
 m\_view.showError("Bad input: '" + firstInput + "' or '" + secondInput + "'");  
 }  
 }  
}

4. Rezultate:

Testarea aplicatiei se face cu ajutorul JUnit. Proiectul contine un folder numit TestDirectory in care se afla clasa ModelTest ce contine metode dupa modelul de mai jos pentru fiecare operatie.

public void polynomialAddition() {  
 Model m = new Model();  
 Polinom p = new Polinom("4x^2+5x^1-2x^0");  
 Polinom q = new Polinom("2x^1+5x^0");  
 m.polynomialAddition(p, q);  
 assertEquals(m.getRezultat(), "4.0x^2+7.0x^1+3.0x^0");  
}

In cazul in care rezultatul furnizat este acelasi cu cel asteptat, testele se executa fara erori.

4. Concluzii:

In concluzie, reprezentarea polinoamelor cu ajutorul ArrayList ajuta intr-o mare masura la implementarea aplicatiei, fiind atat eficienta cat si mai usor de inteles. Interfata grafica reprezinta inca o modalitate de a ajuta utilizatorul sa interactioneze cu aplicatia si sa obtina rezultatele dorite in cel mai simplu mod posibil.

Implementarea ar putea fi dezvoltata prin utilizarea unor functii mai compacte, prin reducerea numarului de instructiuni care realizeaza aceeasi functie, prin notatii care ar reduce cantitatea de cod sau, de asemenea, prin adaugarea de functii si implicit modificarea interfetei care sa ofere mai multe optiuni.

5. Bibliografie:

* <https://www.vogella.com/tutorials/JavaRegularExpressions/article.html>
* <https://www.tutorialspoint.com/junit/junit_basic_usage.htm>
* <https://en.wikipedia.org/wiki/Polynomial_long_division>