**Facultatea de Automatica și Calculatoare**

**Calculatoare și Tehnologia Informației**



**Tehnici de Programare**

**Documentație**

**Tema 1:**

**CALCULATOR POLINOAME**

**Profesor îndrumator: Realizat de:**

**Cristina Pop Câmpean Casiana Ștefana**

**Grupa: 30228**

**Cuprins**

**1. Obiectivul temei**

**2. Analiza problemei, asumptii, modelare, scenarii, cazuri de utilizare, erori**

**3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase,**

**interfete, relatii, packages, algoritmi, interfata utilizator, modul de tratare a erorilor)**

**4. Implementare**

**5. Testare**

**6. Rezultate**

**7. Concluzii**

**8. Bibliografie**

**1. Obiectivul temei**

Propuneti, proiectati si implementati un sistem de procesare a polinoamelor de o singura variabila cu coeficienti intregi.

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Dezvoltarea de use case-uri si scenarii | Realizarea unei diagrame UML și exemplificarea unui scenariu folosind use-case-ul, explicand fiecare pas al scenariului | 2 |
| Alegerea structurilor de date | Alegerea eficientă și potrivită a structurilor de date pentru o mai bună implemetare a proiectului | 3 |
| Impartirea pe clase | Descrierea alegerilor de împarțire pe clase | 3 |
| Dezvoltarea algoritmilor | Descrierea algoritmilor aleși | 3 |
| Implementarea solutiei | Descrierea metodelor, a claselor | 4 |
| Testare | Testare și exemplificare cu JUnit | 5 |

**2. Analiza problemei, asumptii, modelare, scenarii, cazuri de utilizare, erori**



Calculatorul de polinoame poate face 6 operații : adunare, scădere, înmulțire, împărțire, derivare și integrare. Astfel vom obține 6 use-case-uri pentru aceste operații și pe lângă acestea, utilizatorul poate să șteargă câmpurile de text și să introducă sub formă de String datele de intrare pentru polinoame. Pentru use case voi prezenta doar o operație.

Use case: Înmulțire polinoame

Actor: studentul;

Asumpții: Presupunem că datele de intrare pentru cele două polinoame sunt corect scrise.

Scenariu succes :

1. Studentul deschide interfața grafică

2. Studentul introduce corect datele în cele două căsuțe de text pentru polinomul 1 și 2

3. Studentul apasă pe butonul de înmulțire

4. Calculatorul analizează datele introduse și va realiza operația de înmulțire

4. În campul de text pentru rezultat va aparea rezultatul înmulțirii

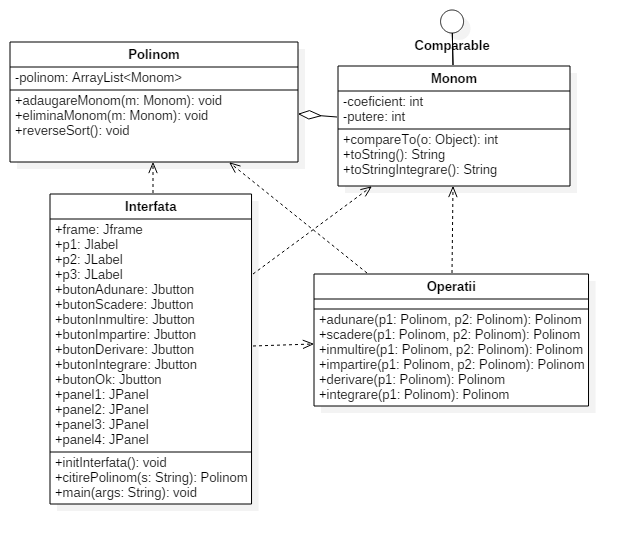
**3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase,**

**interfete, relatii, packages, algoritmi, interfata utilizator, modul de tratare a erorilor)**

**Decizii de proiectare**

Pentru a realiza un sistem de procesare a polinomelor am ales să creez clasa Polinom și Monom. Deoarece un polinom este alcatuit din mai multe monoame, în clasa Polinom voi avea o lisă de monoame, iar în Monom voi avea doar coeficientul și puterea. Deoarece acest sistem are ca rol efectuarea de operații pe polinoame, am ales să creez clasa Operatii unde am implementat toate metodele pentru fiecare operatie. Datorită faptului că acest sistem trebuie folosit de un utilizator, am implementat interfața grafică în altă clasă unde se vor introduce si se vor afișa polinoamele sub formă de String, și tot acolo am implementat metoda necesară interpretării datelor de intrare.

**Diagrama UML**



**Structuri de date**

**S**tructura de dată folosită este ArrayList. Aceasta este folosită pentru a stoca monoamele, fiecare polinom având un ArrayList de monoame.

**Proiectare clase**

Am ales să proiectez patru clase : Monom, Polinom, Interfata și Operatii. Pentru o organizare mai bună a unui polinom, clasa Polinom va conține o listă de monoame, în clasa Monom fiind doar 2 atribute(coeficient și putere). Deoarece calculatorul trebuie să efectueze operații, am ales să realizez clasa Operatii unde am implementat toate metodele pentru realizarea operațiilor, iar pentru interfața grafică am creat o alta clasă unde am realizat interfața și am implementat metoda care tratează datele de intrare.

**Interfete**

Clasa Monom implementează interfața Comparable, care suprascrie metoda compareTo:

@Override

@Override

**public** **int** compareTo(Object o) {

**return** Integer.compare(**this**.putere, ((Monom)o).putere);}

Această metodă compară puterea monomului cu cea a obiectului de la argument, metodă folosită pentru sortarea descrescatoare a ArrayList-ului de monoame după puterea fiecarui monom.

**Relatii**

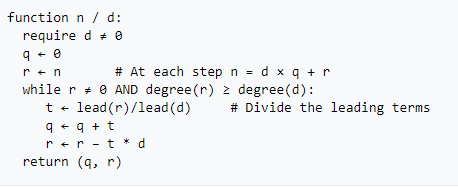
Între clasele programului se pot stabili mai multe relații de legatură :

Între clasa Polinom și monom există o relație de agregare, deaorece Monomul face parte din Polinom, un Polinom conținând mai multe monoame.

Între clasa Operaii și clasele Monom și Polinom există relații de dependență, deoarece în clasa operații se creează obiecte de tip Monom și Polinom. De asemenea, există tot relații de dependență între clasa Interfata și clasele Monom și Polinom tot din aceleași motive.

**Algoritmi**

Unul dintre algoritmii folosiți este cel de la împarțire. Pentru acesta am urmărit urmatorul pseudocod:



Algoritmul implementat de mine este un pic diferit, în loc de un while am folosit un for și un if.

**4. Implementare**

Proiectul conține 4 clase :

* Monom
* Polinom
* Operatii
* Interfata
* TestOperatii

**Clasa Monom**

Aceasta clasa alcatuieste un monom, fiecare monom avand 2 atribute: gradul și puterea.

Pentru aceasta clasă am implementat mai întai un constructor având parametrii coeficient și putere, care inițializează monomul cu ce valori dorim. În continuare sunt implementate setterele si getterele pentru cele două atribute, cu ajutorul cărora vom putea sa facem operațiile necesare.

Urmatoarea metodă este metoda suprascrisă compareTo, având ca parametru Object o, care compară puterea monomului cu cea puterea obiectului dat ca și parametru, metodă folosită pentru ordonarea monoamelor în lista de monoame din polinom.

Ultimele două metode implementate sunt toString(metodă suprascrisă) și toStringIntegrare. Acestea vor returna un alt String care va reprezenta scrierea monomului în funcție de coeficientul si puterea sa, ultima metodă comparativ cu prima este folosită doar în momentul în care folosim operația de integrare, metoda concatenând numitorul rezultatului dupa integrare. Cele două metode vor fi utilizate pentru afișarea corectă a polinomului rezultat în urma operațiilor.

**Clasa Polinom**

Acesta clasă conține un ArrayList de monoame. In aceasta am implementat mai întai settere și gettere cu ajutorul cărora se vor putea realiza operațiile pe aceste liste de monoame. Urmatoarele două metode, adaugareMonom și eliminaMonom vor adauga sau șterge un monom din ArrayList, metode esențiale pentru realizarea operațiilor pe polinoame. Ultima metodă este reverseSort, care cu ajutorul metodei compareTo implementată în clasa Monom, va sorta ArrayList-ul după puterea fiecarui monom. Această metodă ajută la aranjarea fiecarui monom după puterea sa, rezultatul fiind scris și afișat în ordine corectă, cea mai mare putere fiind prima.

Clasa Operatii

În această clasă sunt implementate toate operațiile cerute pe polinoame: adunare, scădere, înmulțire, împarțire, derivare și integrare.

Primele două metode sunt realizate aproape la fel, ambele primesc ca și argumente doua polinoame p1 și p2 de tip Polinom si returneaza un rezultat de tip Polinom. Fiecare dintre cele două polinoame sunt parcurse cu un for, iar daca puterea monoamelor din cele două liste este aceeași atunci acestea se adună sau se scad, iar monomul de la poziția curentă din cel de-al doilea polinom se sțerge din listă. La finalul celor două for-uri, cu ajutorul lui foreach vom parcurge elementele rămase din cel de-al doilea polinom și le vom adauga la polinomul final. În cazul scăderii, pentru elementele rămase în cel de-al doilea polinom, coeficienții acestora se vor inmulți cu -1, pentru a da rezultatul corect în urma scăderii.

Pentru metodele de împarțire și înmulțire se primesc tot ca și argumente două polinoame și se returnează la fel tot un alt polinom. În metoda înmulțirii parcurgem mai întai cele două polinoame cu foreach, adăugăm în pRez fiecare monom înmulțit cu fiecare monom din cele două liste, după care urmând să parcurgem cu două for-uri polinomul rezultat și să verificăm dacă în acesta se gasesc elemente cu aceeași putere, dacă da să le adunam la elementul de pe prima poziție și să ștergem elementul de pe a doua poziție.

La metoda împarțirii am utilizat ideea de bază a algoritmului de împarțire, prezentată în capitolul anterior prin pseudocod, înlocuind while-ul cu un for și un if.

Ultimele două metode, derivarea si integrarea au un singur argument de tip Polinom și vor returna de asemenea un rezultat de tip Polinom. Ambele metode schimba valoarea coeficienților și a puterilor, dar la integrare am ales să nu impart coeficientul cu puterea+1, ci să implementez metoda toStringIntegrare din clasa Monom pentru a evita utilzarea de date double și pentru o afișare a rezultatului cu un aspect mai aproape de cel folosit în scris.

**Clasa Interfata**

Aceasta clasa conține trei metode foarte importante: initInterfata() care realizeaza interfața GUI, citirePolinom care returnează un polinom după ce primește ca și argument un String și funcția main.

Mai întâi am declarat în clasă toate componentele necesare realizării interfeței:

-frame de tip JFrame care va fi fereastra principală a programului,

-trei Jlabel

-trei JTextField, unde se introduc cele doua polinoame și se afișează rezultatul final

7 butoane Jbutton, pentru cele 6 operații și încă un buton pentru ștergerea textului

4 paneluri JPanel,prin care vom grupa toate aceste elemente.

În metoda initInterfata am creat toate componentele necesare. Am dat numele frame-ului, mărimea sa, am setat să fie vizibilă și am setat sa nu se poată modifica marimea(frame.setResizable(false);) și am setat Layout-ul de tip GridLayout(4, 1). La frame am adugat 4 panel-uri. La acestea le-am setat Layout-ul de tip FlowLayout. Fiecare Panel va conține un grup de componente. Primul conține Labelul numit Polinom 1, după care va fii un JtextField unde vom introduce primul polinom. Al doilea panel conține label-ul și JtextField pentru cel de-al doilea polinom. Urmatorul panel este alcatuit din cele 7 butoane, iar ultimul conține labelul si JtextField pentru rezultat. La aceste patru panel-uri le-am modificat culoarea background-ului folosind metoda panel1.setBackground(new Color(35, 2, 49)); și le-am facut vizibile. Alte modificari de design sunt prezente și la butoane sau la labeluri, unde am modificat marimea, fontul sau backgroundul.

Pentru fiecare buton am folosit metoda addActionListener, cu ajutorul careia voi putea afișa rezultatul dorit. Fiecare buton are o alta implementare, în funcție de tipul operației. Pentru butonAdunare am creat un ActionListener unde am suprascris metoda actionPerformed. În aceasta am inițializat trei polinoame, o operație și un String rez. Primele doua polinoame vor primi cu ajutorul metodei citirePolinom valorile pe care noi le scriem în cele două JtextField Polinom1 și Polinom2 (pol1=citirePolinom(polinom1.getText());). În continuare vom face operația de adunare(polRez=op.adunare(pol1,pol2);), vom aranja lista de monoame rezultată dupa putere(polRez.reverseSort();), iar la final cu ajutorul unui foreach vom converti valoarea monoamelor din listă în String(rez=rez+m.toString();) și vom afișa în JtextField-ul Rezultatului ce am obținut(polinomRez.setText(rez);). Pe aceeași idee sunt implementate și celelalte addActionListener, diferențe fiind la înmulțire(dacă unul dintre polinoame este 0 rezultatul va fii 0), împărțire(dacă al 2=lea polinom este 0 se afișează o eroare, iar dacă primul este 0, rezultatul va fii 0), derivare(dacă polinomul are puterea 0 rezultatul va fii 0), integrare(convertirea valorilor din lista de monoame în String se realizeză cu metoda toStringIntegrare()). Cel de-al 7-lea buton este butonOk care realizează stergerea tuturor elementelor JtextField.

Urmatoarea metodă este una esențială. Metoda ciirePolinom primeste ca și argument un String și returnează un rezultat de tip Polinom. Această metoda a fost implementată cu Regex, site-urile de unde am învațat fiind incluse in bibliografie.

“O expresie regulată, regex este, în informatica teoretică⁠ și în teoria limbajelor formale, un șir de caractere⁠ care definesc un șablon⁠ de căutare. De obicei, acest șablon este apoi utilizat de către algoritmii de căutare pe șiruri⁠ pentru operațiuni de „căutare” sau „căutare și înlocuire” operațiuni pe șiruri de caractere⁠.”

Pentru început am declarat un nou polinom care va conține rezultatul final. Stringul care îmi va detecta pattern-ul polinomului este urmatorul:

String Regex = "((-?\\d+(?=x))?(-?[xX])(\\^(-?\\d+))?)|(-?\\d+)";

Acest String respecta convențiile Regex-ului, fiind alcatuit din 6 grupuri. Fiecare grup tratează mai multe caractere ca un tot, și fiecare grup reprezintă un anumit caz în care se poate afla un monom.

-group(1) = ((-?\\d+(?=x)) ->existenta unui termen valid

-group(2) = (?=x) ->valoarea coeficientului

-group(3) = (-?[xX]) ->existenta lui x

-group(4) = (\\^(-?\\d+)) ->existenta unei puteri

-group(5) = (-?\\d+) ->valoarea puterii

-group(6) = (-?\\d+) ->valoarea coeficientului lui x^0

Pentru a putea forma polinomul final, vom folosi un while care se va executa atata timp cât va gasi un meci intre Stringul de la argument si patternul dat de Regex.

În acest while voi avea 5 cazuri, iar în funcție de fiecare caz, voi construi monomul care va fi adăugat listei de monoame din rezultatul final.

CAZUL 1: 2x^3, -2x^3 ;coeficient = group(2) , putere = group(5);

CAZUL 2: x^2, -x^2 ; coeficient = 1 / -1 , putere = group(5);

CAZUL 3: 2x, -2x ; coeficient = group(2), putere = 1;

CAZUL 4: x, -x ; coeficient = 1 / -1, putere = 1;

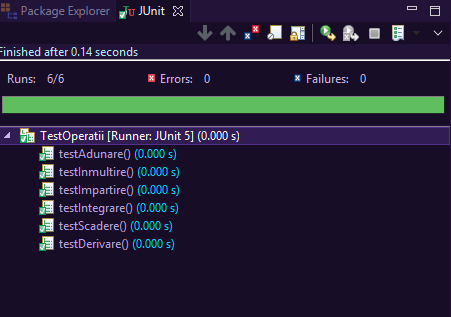
CAZUL 5: 2, -2 ; coeficient = group(6), putere = 0;

**5. Testare**

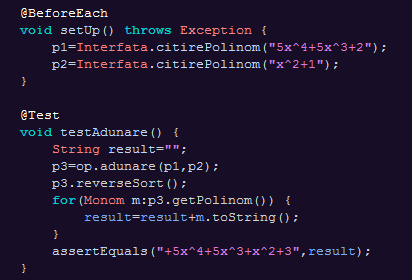
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Date intrare | Rezultat asteptat | Rezultat obținut | Pass/Fail |
| Adunarea | p1=Interfata.citirePolinom("5x^4+5x^3+2");  p2=Interfata.citirePolinom("x^2+1"); | "+5x^4+5x^3+x^2+3" | "+5x^4+5x^3+x^2+3" | Pass |
| Scaderea | "+5x^4+5x^3-x^2+1" | "+5x^4+5x^3-x^2+1" | Pass |
| Inmultirea | "+5x^6+5x^5+5x^4+5x^3+2x^2+2" | "+5x^6+5x^5+5x^4+5x^3+2x^2+2" | Pass |
| Impartirea | "+5x^2+5x-5" | "+5x^2+5x-5" | Pass |
| Derivarea | "+20x^3+15x^2" | "+20x^3+15x^2" | Pass |
| Integrarea | "+5x^5/5+5x^4/4+2x" | "+5x^5/5+5x^4/4+2x" | Pass |

Testarea se realizează în clasa TestOperatii unde sunt verificate toate operațiile.

**6. Rezultate**

****

Toate operațiile testate au avut un rezultat pozitiv.



Pentru ca testele să aibă rezultatul așteptat, toate meodele folosite trebuie să fie corecte, astfel toate aceste metode ( citirePolinom, adunare, reverseSort, toString ) sunt realizate corect, cu ajutorul lor se efectuează interpretarea datelor, transformarea lor în polinoame, efectuarea operațiilor și transformarea înapoi a Polinoamelor în Stringuri.

**7. Concluzii**

În concluzie, acest prim proiect m-a determinat să aplic și să îmi amintesc teoria și tehnicile de programare învățate în semestrul trecut, să realizez o interfață grafică de la zero, fără a folosi tool-uri de drag and drop. De asemenea mi-a dezvoltat o atenție mai sporită pentru evitarea greșelilor și pentru detectarea acestora.

Există mai multe dezvoltări ulterioare a programului, precum:

-validarea datelor de intrare

-implementarea a mai multor operații mai complexe

-realizarea mai complexă a interfeței grafice

**8. Bibliografie**

1. <http://www.mkyong.com/tutorials/junit-tutorials/>

2. <http://tynerblain.com/blog/2007/04/09/sample-use-case-example/>

3. <https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>

4. <http://zetcode.com/tutorials/javaswingtutorial/>

5. <https://en.wikipedia.org/wiki/Polynomial_long_division>

6. <https://www.tutorialspoint.com/java/java_regular_expressions.htm>

7. <https://en.wikipedia.org/wiki/Regular_expression>

8. <https://stackoverflow.com/questions/28859919/java-regex-separate-degree-coeff-of-polynomial>