*Tehnici de programare Universitatea Tehnică din Cluj -Napoca*

*Tema 1*

**Calcul de polinoame**

**-documentație-**

*Nimigean Emanuela*

*CTI, Grupa 30226*

♦Cerințe funcționale

Dezvoltarea unui calculator de polinoame care să permită efectuarea următoarelor operații:

* Citirea polinoamelor de la tastatură sub forma: -2\*x^3+4\*x^5-1\*x^1+10 (

-monoamele pot fi introduse în orice ordine, acestea urmând să fie sortate înainte de afișarea polinomului rezultat în urma efectuării operației dorite

-polinomul trebuie introdus cu toți coeficienții/exponenții săi nenuli, inclusiv „1”, pentru a asigura funcționarea corectă a calculatorului de polinoame)

* Adunare
* Scădere
* Înmulțire
* Împărțire
* Derivare
* Integrare
* Obiectivul temei

Obiectiv principal: propunerea, proiectarea și implementarea unui sistem de procesare a polinoamelor de o singură variabilă, cu coeficienți întregi.

Obiective secundare:

* Dezvoltarea de use-case -uri și scenarii -detalii în cap. următor
* Împărțirea pe clase : proiectul este structurat pe cinci clase, împărțite în trei pachete astfel: clasele Monom și Polinom sunt incluse în pachetul model, clasa Operatii este inclusă în pachetul operations, iar clasele Frame și Panou sunt incluse în pachetul view -detalii în cap . Proiectare și Implementare
* Implementarea soluției -detalii în cap . Implementare
* Testare -detalii în cap . Rezultate
* Analiza problemei, modelare, scenarii, cazuri de utilizare

Se cere implementarea unui calculator de polinoame care să permită efectuarea următoarelor operații:

-Citirea polinoamelor de la tastatură sub forma: -2\*x^3+4\*x^5-1\*x^1+10 (

-monoamele pot fi introduse în orice ordine, acestea urmând să fie sortate înainte de afișarea polinomului rezultat în urma efectuării operației dorite

-polinomul trebuie introdus cu toți coeficienții/exponenții săi nenuli, inclusiv „1”, pentru a asigura funcționarea corectă a calculatorului de polinoame)

- Adunare

- Scădere

- Înmulțire

- Împărțire

- Derivare

- Integrare

Use case name : Polynomial Calculator

Actors :

Triggers : user-ul indică faptul că dorește efectuarea unei anumite operații pe polinomul / polinoamele introduse

Preconditions : user-ul a introdus polinomul / polinoamele pe care vrea să le folosească

Post-conditions :

- programul va citi polinomul / polinoamele

- va fi selectată și efectuată operația cerută

- se furnizează un rezultat

Normal flow :

- user-ul introduce polinomul / polinoamele

- user-ul selectează operația dorită prin apăsarea butonului corespunzător acesteia

- polinoamele sunt citite și sunt validate de către parser și transpuse în obiecte de clasă Polinom

- se apelează metoda corespunzătoare operației alese pe aceste polinoame date ca parametru

- se calculează rezultatul operației

- polinomul rezultat este afișat în text field

- dacă se dorește efectuarea unei alte operații , use case-ul se reia

Alternate flows :

1. - user-ul introduce polinomul / polinoamele

- user-ul selectează operația dorită prin apăsarea butonului corespunzător acesteia

- polinoamele sunt citite, însă nu respectă formatul impus, deci nu pot fi parsate

- use case-ul se reia

1. - user-ul introduce doar un polinom

- user-ul selectează una din operațiile care necesită două polinoame ( adunare , scădere , înmulțire )

- programul nu găsește al doilea polinom și use case-ul se reia

1. - user-ul introduce polinomul ce dorește să fie derivat sau integrat în al doilea text field, nu în primul- cum se cere

- programul nu găsește polinomul și use case-ul se reia

1. - user-ul introduce polinoamele

-user-ul resetează panoul

-use case-ul se reia

5. - user-ul introduce polinoamele

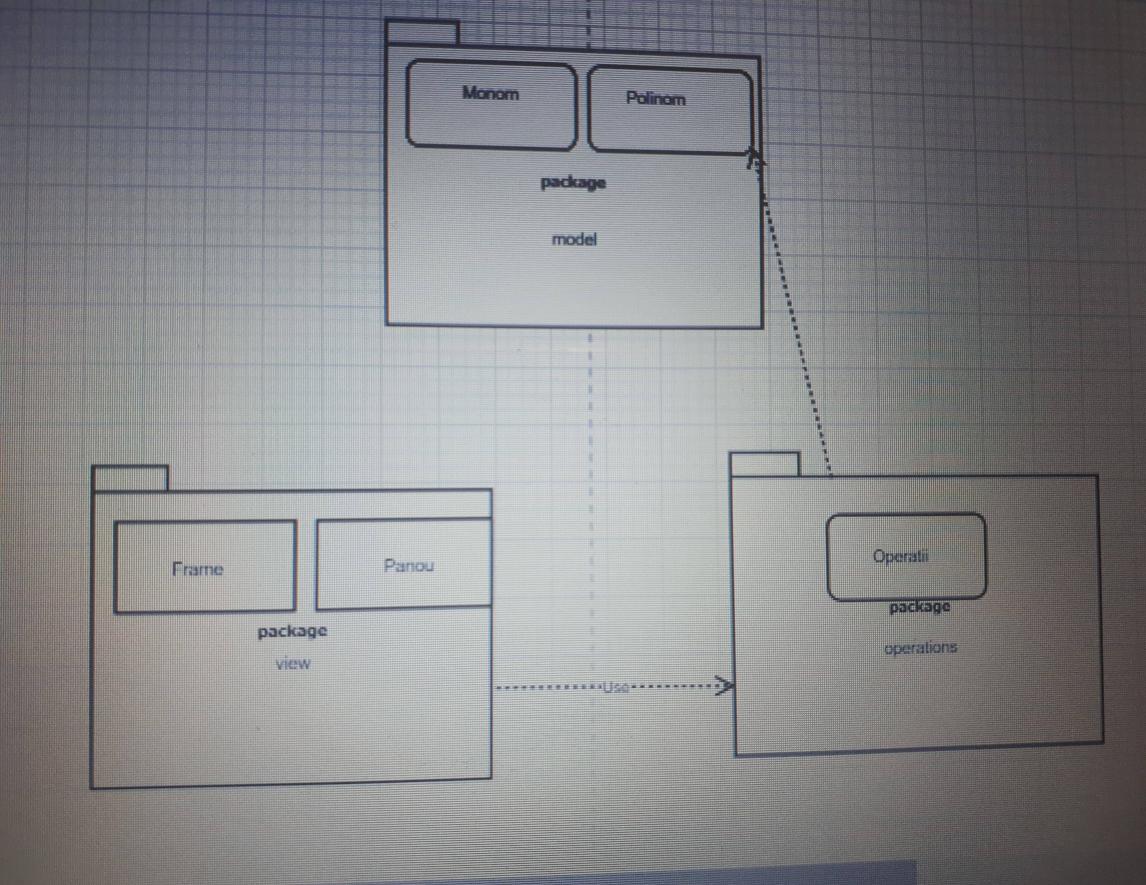
-user-ul închide fereastra

-use case-ul se încheie

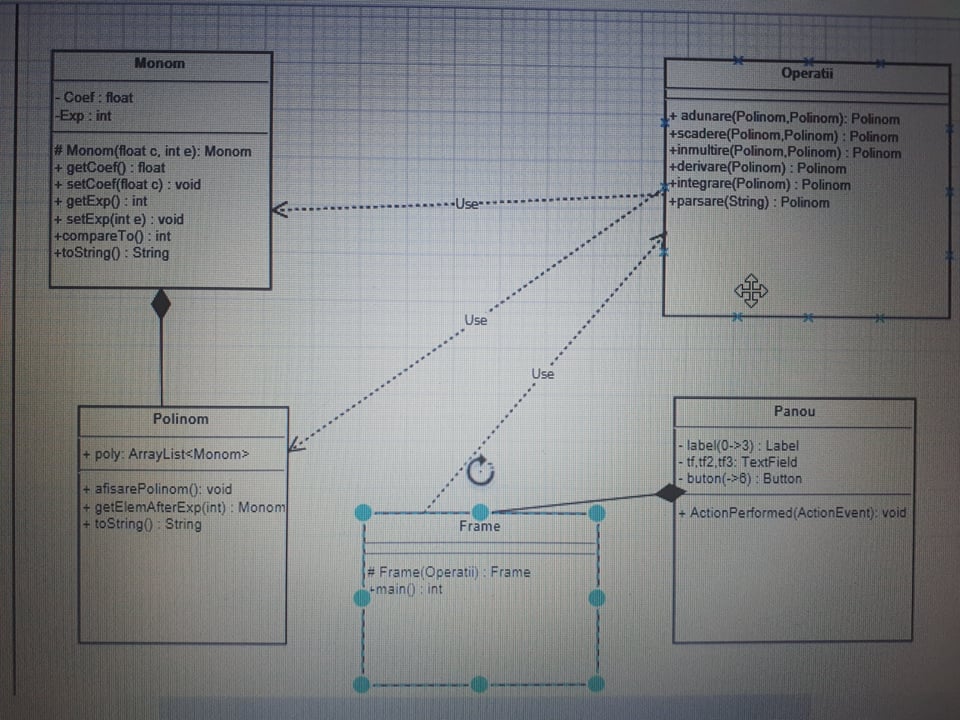
Actorul, cel ce va interacționa cu acest Calculator de Polinoame este reprezenat de utilizator care, pentru a efectua operații pe polinoame, apelează la acest program .

* Proiectare ( decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator )

Împărțirea proiectului pe pachete am făcut-o separând cele trei componente ale acestuia: modelul ( package model - clasa Monom și clasa Polinom ) , interfața (package view - clasa Frame și clasa Panou ) și operațiile efectuate ( package operatii ).



În ceea ce privește clasele, cele trei pachete conțin cinci clase : Monom, Polinom, Operatii, Frame, Panou .



Clasa Frame conține o instanță a clasei Operații, clasa Operatii conține la rândul ei instanțe ale claselor Monom și Polinom, iar Panoul conține în cadrul ascultătorilor săi instanțe ale claselor Operatii, Monom și Polinom .

Interfața utilizator este una simplă, clasică și ușor de utilizat. Compusă dintr-o fereastră în cadrul căreia am pus un panou cu elementele sale ( etichete, butoane, text fields ), permite o interacțiune facilă a utilizatorului cu programul ce efectuează operații pe polinoame .

* Implementare

Proiectul este structurat in 5 clase , după cum urmează:

- 2 clase aparținând modelului : clasa Monom și clasa Polinom

- 2 clase aparținând interfeței cu utilizatorul : clasa Frame și clasa Panou

- o clasă în care am definit operațiile cu polinoamele definite cu ajutorul claselor Monom și Polinom : Operații .

**Clasa Monom** descrie monoamele din cadrul unui polinom ( care sunt de forma coef \* x ^exp ), având ca atribute coeficientul și exponentul monomului . Coeficientul este definit ca număr real pentru a putea fi folosit și pentru a da rezultatele corecte în cadrul operației de integrare, unde coeficientul este împărțit la noul exponent. Contructorul cu parametri explicit permite construirea unui monom cu coeficientul și exponentul dat de noi . Am implementat și metode accesoare și mutatoare pentru a putea avea acces la coeficientul și exponentul monomului și a-l putea modifica, altfel fiind imposibil, atributele fiind declarate private . Am suprascris metoda toString() în această clasă pentru a putea afișa monoamele în cadrul unui polinom, având în vedere anumite situații speciale de monom cum ar fi cel cu exponent 0 care este de fapt o constantă, dar am controlat afișarea și în funcție de coeficientul monomului ( dacă este pozitiv, respectiv negativ sau dacă este egal cu +1 / -1 ).

Ultima metodă este compareTo(Monom m) pe baza căreia am implementat metoda de sortare descrescătoare a monoamelor dintr -un polinom, in clasa Polinom. Aceasta compară monoamele după exponent și returnează valoarea 0 daca au același exponent, -1 dacă primul monom are exponentul mai mare, respectiv 1 dacă monomul dat ca parametru în metodă are exponentul mai mare .

**Clasa Polinom** , bazându -se pe clasa Monom, descrie un polinom, având ca atribut un ArrayList de monoame, în care apar toate monoamele dintr-un polinom . Am suprascris și aici metoda toString, alegând să afișez polinoamele folosindu -mă de metoda toString deja suprascrisă în clasa monom . Metoda de afișare , folosită în principal pe parcursul implementării proiectului , afișează pur și simplu toate monoamele , unul după altul .

Metoda getElemAfterExp ( int e ) primește un număr întreg și returnează monomul din polinomul curent care are exponentul egal cu valoarea dată prin parametru , în cazul în care acesta există în polinomul dorit .

**Clasa Operații** , depinzând de cele două clase anterioare ( Monom și Polinom ), implementează cinci dintre cele șase operații cerute, respectiv : adunare, scădere, înmulțire, derivare și integrare , dar și parsarea unui polinom introdus de utilizator sub formă de string.

Metoda adunare(Polinom p1 , Polinom p2) realizează adunarea polinoamelor primite ca parametru , urmând ca noul polinom rezultat să conțină atât monoamele din primul polinom primit ca parametru cât și cele din al doilea polinom primit ca parametru . Se parcurge folosind o instrucțiune repetitivă cu număr cunoscut de pași „for” primul polinom , salvăm coeficientul și exponentul monomului curent , verificăm dacă și în cel de -al doilea polinom există un monom cu același exponent folosindu -ne de metoda getElemAfterExp din clasa Polinom, iar dacă există îl adunăm cu monomul curent și îl adăugăm apoi în polinomul rezultat , stergându-l totodată din al doilea polinom . Altfel, dacă nu există în polinomul al doilea un monom cu acest exponent , adăugăm monomul curent direct în polinomul rezultat . Tot în acesta vom adăuga la final restul monoamelor rămase în cel de -al doilea polinom . În cele din urmă , sortăm monoamele din polinomul final pentru o mai usoară percepere a acestuia de către utilizator .

Metoda scădere ( Polinom p1 , Polinom p2 ) este bazată tot pe metoda anterioară ce realizează adunarea a două polinoame , având în plus însă faptul că înainte de a le aduna, cel de-al doilea polinom este înmulțit cu -1 .

Metoda înmulțire ( Polinom p1 , Polinom p2 ) realizează înmulțirea celor două polinoame primite ca parametru, în 3 pași : mai întâi , parcurgem cele două polinoame, înmulțind fiecare

monom din primul polinom cu fiecare monom din cel de-al doilea polinom și adăugând monoamele rezultate în urma acestor înmulțiri în polinomul rez . Facem o copie a polinomului rez într -un alt polinom , rez\_final în care vom avea în final rezultatul înmulțirii celor două polinoame , însă cu un singur monom de fiecare grad .

Metoda derivare ( Polinom p1 ) calculează derivata polinomului primit ca parametru aplicând formula de derivare :

( c\*x ^e ) ’= e\*c\*x ^( e-1 ) . Însă este tratat separat cazul în care un monom din cadrul polinomului primit are exponentul 0 adică este o constantă, derivata acestuia fiind 0, iar in field-ul corespunzător rezultatului va rămâne gol .

Metoda integrare ( Polinom p1 ) integrează polinomul primit ca parametru folosind formula de integrare :

( c\*x ^e ) ’ = [ c\*x ^( e+1 )] /( e+1 ) .

Metoda parsare ( String input ) realizează parsarea String -ului introdus de utilizator, adică transformarea lui din String într-un obiect de clasă Polinom . Mai întâi, fiecare semn ‘-’ este înlocuit cu ‘+-’ pentru ca mai apoi să se poată face împărțirea String -ului în bucăți folosind delimitatorul ‘+’. Dacă String -ul inițial începea cu ‘-’ , asta înseamnă că după înlocuire va începe cu ‘+-’ , lucru ce va determina o funcționare eronată a funcției split (”\\+”) deoarece aceasta se așteaptă să fie ceva și de o parte și de cealaltă a lui ‘+’, fapt ce nu va fi adevărat în acest caz . Așadar, pentru a evita această funcționare greșită, am ales să șterg primul caracter din String după înlocuire , însă doar dacă acesta este ‘+’. Folosim split() de două ori, mai intâi cu parametrul ‘+’, iar apoi cu ‘\*x^’ . Astfel , rămânem cu coeficientul și exponentul fiecărui monom , pe baza cărora îl vom și construi și apoi adăuga în polinomul rezultat .

Toate metodele au la final folosită metoda de sortare a monoamelor pentru ca polinomul ce rezultă în urma operației respective să fie mai lizibil, mai ușor de înțeles, mai ușor de refolosit .

Pentru interfața utilizator, am ales împărțirea ei în doua clase și anume : clasa Frame ce creează fereastra ce va servi în comunicarea cu utilizatorul și clasa Panou ce creează conținutul atribuit ferestrei, cu toate componente sale ( butoane, text fields, etichete ) . **Clasa Frame** conține și metoda main() din care se instanțiază un obiect de tipul clasei Operații și unul de tipupl clasei Frame .

**Clasa Panou** creează panoul ce va fi asignat ferestrei create în clasa Frame. În cadrul acesteia, am declarat mai multe etichete ( label0, label, label2, label3 ), text fields ( tf, tf2, tf3 ), precum și butoane corespunzând fiecărei operații implementate . Am adăugat componentele enumerate anterior panoului , iar apoi am implementat ascultători pentru butoanele folosite . În cadrul acestora se face citirea polinoamelor din text fields , apelarea metodelor corespunzând operațiilor, respectiv parsării , dar și afișarea în text field-ul corespunzător rezultatului operației alese .

* Rezultate

Prin testarea calculatorului pentru mai multe variante de intrare , am verificat funcționarea acestuia atât la introducerea unor forme neacceptate de intrare , precum și a celei permise . Polinomul trebuie să respecte structura impusă și anume : Coef\*x^Exp+/-Coef\*x^Exp , fiind necesară introducerea tuturor coeficienților și exponenților nenuli, inclusiv cazul în care unul din ei este „1”, fără spații între elemente și fără ‘+’ la începutul polinomului .

Caz favorabil (= toți coeficienții/ exponenții și nu începe cu ‘+’ ) : p1 = -2\*x^3+4\*x^1-1\*x^5+10 , p2 = 2\*x^3-4 . Atunci : p1+p2 = -x^5+4.0\*x^1+6.0 , p1-p2 = -x^5-4.0\*x^3+4.0\*x^1+14.0, p1\*p2 = -2.0\*x^8-4.0\*x^6+4.0\*x^5+8.0\*x^4+28.0\*x^3-16.0\*x^1-40.0 , p1( derivat ) = -5.0\*x^4-6.0\*x^2+4.0 , p1 (integrat ) = -0.16666667\*x^6-0.5\*x^4+2.0\*x^2+10.0\*x^1 .

Încercând și exclusiv cu constante : p1 = 23 , p2 = 62 . Atunci p1+p2 = +85.0 , p1-p2 = -39.0 , p1\*p2 = +1426.0 , p1( deriv ) =(null) , p1( integ ) = +23.0\*x^1 .

Cazul nefavorabil ( încălcarea uneia dintre regulile menționate mai sus ) duce la nerezolvarea parsării polinomului introdus și prin urmare , inexistența unei soluții .

* Concluzii

Implementarea unui program care realizează efectuarea operațiilor pe polinoame prezintă mai multe mici dificultăți / provocări întâmpinate în special în cazul lucrului cu obiecte, nu doar cu simple numere . Totuși, tema aceasta contribuie semnificativ la exersarea atât a limbajului de programare Java, cât și a conceptelor ce țin de programarea orientată pe obiect .

Ca dezvoltare ulterioară, calculatorului i se pot adăuga și alte funcții mai complexe precum ridicarea la putere sau altele și i se poate realiza o interfață mai complexă.

* Bibliografie

-www.coned.utcluj.ro/~salomie/PT\_Lic/

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

-http://users.utcluj.ro/~igiosan/teaching\_poo.html