**CUPRINS:**

* Obiectivul temei
* Analiza problemei, modelare, scenarii, cazuri de utilizare
* Proiectare
* Implementare
* Rezultate
* Concluzii
* Bibliografie

***1) Obiectivul temei***

Obiectivul acestei teme este de a crea un calculator pentru efectuarea operațiilor de adunare, scădere, înmulțire, împărțire, derivare și integrare a polinoamelor.

Obiective secundare:

* Crearea clasei pentru un monom = pentru a ține minte puterea monom
* Crearea claselor MonomialInt si MonomialDouble = pentru a tine minte coeficientul intreg respectiv de tip double al unui monom (mostenesc clasa Monomial)
* Crearea clasei pentru un polinom = definit ca o succesiune de adunări ale monoamelor
* Crearea unei clase de tip Comparator = pentru a sorta descrescator monoamele dintr-un polinom dupa gradul acestora
* Crearea modelului pentru MVC = creare ajutată de clasa Polinom și implicit de clasa Monomial
* Crearea view-ului pentru MVC = creare stric pentru interfața utilizator
* Crearea controlului pentru MVC = unde se execută fiecare opțiune a calculatorului

Toate aceste obiective vor fi detaliate în capitolul 4.Implementare

***2) Analiza problemei, modelare, scenarii, cazuri de utilizare***

In aceasta problema se pot aduna,scadea,inmulti si imparti doua polinoame in timp ce derivarea si integrarea se face numai pe un singur polinom.

a) Descrierea use-case pentru adunare,scadere,impartie,inmultire:

* Se deschide aplicatia
* Se ruleaza metoda main din clasa Controller aflata in pachetul Controller
* Se introduce primul polinom in primul textField si al doilea polinom in al doilea textField
* Se apasa pe butonul de + pentru adunare, - pentru scadere, X pentru inmultire, / pentru impartie
* Se apasa pe butonul de egal =
* Se observa rezultatul in al treilea textField

b) Descrierea use-case pentru derivare si integrare

* Se deschide aplicatia
* Se ruleaza metoda main din clasa Controller aflata in pachetul Controller
* Se introduce NUMAI primul polinom in primul textField
* Se apasa pe butonul de Derivare pentru derivare, Integrare pentru integrare
* Se apasa pe butonul de egal =
* Se observa rezultatul in al treilea textField

***3) Proiectare***

Diagrama UML:

A picture containing text, electronics

Description automatically generated

* Prima decizie pe care am luat-o a fost aceea că inputul utilizatorului trebuie să respecte unele reguli când vine vorba de introducerea polinomului și anume:

1) dacă se dorește introducerea unui coeficient negativ atunci se va pune semnul - fix înainte de numar (ex. 2\*x^3 + -4\*x^5)

2) dacă se dorește introducerea numai a unui coeficient atunci puterea lui x trebuie sa fie 0 (ex. vreau să introduc numai 4 atunci scriu 4\*x^0)

3) coeficientii nu au voie sa fie numere reale

4) la operatiile de adunare, scadere, inmultire, impartire se introduc ambele polinoame pe cand la derivare si integrare doar primul

* După această primă decizie am creat clasa Monomial care are o singura metoda care verifica daca doua monoame au aceeasi putere.
* Dupa am creat clasa MonomialInt si MonomialDouble ce mostenesc clasa Monomial. In clasa MonomialInt am creat majoritatea metodelor ce efectueaza operatiile mentionate anterior pe monoame in timp ce clasa MonomialDouble are numai metoda de a inmulti doua monoame (unul de tipul MonomialDouble si celalalt fiind de tipul MonomialInt)
* După am creat clasa Polinom ce conține un ArrayList de tipul Monomial pentru a simplifica operațiile între două polinoame ca fiind mai multe operații între monoamele ce alcătuiesc polinoamele.
* Clasa Model are două polinoame și metode ce apelează operațiile dintre acestea. Clasa View conține tot ce ține de interfața utilizator, iar clasa Controller face legătura dintre ea, Model și View pentru a crea MVC-ul.
* Algoritmul ce trebuie explicat este cel de impartie a doua polinoame.

Acest algoritm are urmatorii pasi:

1. Monomul cu cel mai mare grad din deimpartit se imparte la cel mai mare monom al impartitorului
2. Rezultatul se adauga in cat
3. Se creaza un nou polinom ce este rezultatul inmultirii dintre rezultatul obtinut la pasul 2 si impartitorul
4. Deimpartitul devine rezultatul scaderii dintre el insusi si polinomul obtinut la pasul 3
5. Se reia algoritmul de la inceput cat timp gradul celui mai mare monom din deimpartit este mai mare sau egal decat gradul celui mai mare monom din impartitor

***4) Implementare***

* Clasa Monomial:

1. Ca variabila instanta este puterea unui monom stocat de tip private int.

2. Am un constructor pentru aceasta variabila instanta.

3. Am un getter pentru a putea lua puterea.

Am o singura metoda pe langa cele prezentate mai sus in care se verifica daca doua monoame au aceeasi putere ( folositoare pentru adunarea si scaderea lor)

* Clasa MonomialDouble:

1. Aceasta clasa mosteneste clasa Monomial.

2. Ca variabila instanta am coeficientul unui monom de tip private double (in cazul in care doua numere se impart exact este nevoie ca rezultatul sa fie cu virgula flotanta).

3. Constructorul e pentru a initializa coeficientul si puterea unui monom ce are coeficient real.

4. Metoda multiplyMonomials are ca parametru un monom de tipul MonomialInt (coeficientul lui este real) si returneaza un monom de tipul MonomialDouble dupa ce se face inmultirea coeficientilor acestora si adunarea puterilor.

5. Am doua gettere atat pentru puterea cat si pentru coeficientul monomului.

* Clasa MonomialInt:

1. Aceasta clasa mosteneste clasa Monomial.

2. Ca variabila instanta am coeficientul unui monom de tip private int (pentru adunare, scadere, inmultire si derivare a unui monom)

3. Constructorul e pentru a initializa coeficientul si puterea unui monom ce are coeficient intreg.

4. Am doua gettere pentru coeficient si putere.

5. Metodele additionMonomials, substractMonomials, multiplyMonomials si derivateMonomial toate returneaza un monom de tip MonomialInt dupa efectuarea operatiilor de adunare, scadere, inmultire si derivare (primele trei metode au si ca parametru un alt monom de tip MonomialInt in timp ce ultima metoda nu pentru ca derivarea se face pe un singur monom).

* In **additionMonomials** se verifica daca, mai intai monoamele au aceeasi putere (pentru ca adunarea se poate face doar daca monoamele au aceeasi putere), iar daca nu au se returneaza null. Daca conditia este adevarata atunci urmeaza sa se verifice daca coeficientii sunt opusi deoarece daca o sa se adune coeficientul o sa fie 0, iar un monom cu coeficient 0 este 0. Daca coeficientii nu sunt opusi atunci se returneaza un MonomialInt cu coeficientul fiind suma coeficientilor si aceeasi puterea cu cea a celor doua monoame. Daca coeficientii sunt opusi atunci se returneaza null.
* In **substractMonomials** se verifica mai intai daca cele doua monoame au aceeasi putere. Daca nu se returneaza null, daca da atunci se trece la urmatoare conditie de a verifica daca scaderea coeficientilor celor doua monoame are rezultatul 0. Daca da se returneaza null, daca nu atunci se returneaza un MonomialInt cu coeficientul fiind scaderea dintre cei doi coeficienti si puterea este aceeasi cu cea a celor doua monoame.
* In **multiplyMonomials** se returneaza un MonomialInt care are ca coeficient inmultirea coeficientilor celor doua monoame si puterea este suma puterilor celor doua monoame.
* In **derivateMonomial** se returneaza un MonomialInt cu coeficientul fiind rezultatul inmultirii dintre coeficientul monomului si puterea acestuia, iar puterea este puterea monomului decrementata cu 1

Metodele divideMonomials si integrateMonomial returneaza un monom de tip Monomial deoarece nu stim daca in urma operatiilor coeficientul poate sa ramana de tip intreg sau de tip real. (returnam fie MonomialInt, fie MonomialDouble)

* In **divideMonomials** verificam daca restul impartirii coeficientilor da 0. Daca da o sa se returneze in MonomialInt cu coeficientul fiind catul impartirii dintre cei doi coeficienti ale monoamelor si puterea este rezultatul scaderii celor doua puteri ale monoamelor. Daca restul nu da 0, atunci se returneaza un MonomialDouble cu coeficientul si puterea exact ca in cazul in care returnam un MonomialInt.
* In **integrateMonomial** se verifica in prima faza daca restul impartirii dintre coeficient si putearea incrementata cu 1 este 0 sau nu. Daca restul este 0 atunci se returneaza un MonomialInt ce are ca coeficient catul impartirii dintre coeficient si puterea incrementata cu 1, iar puterea este doar incrementata cu 1. Daca restul nu este 0 atunci se returneaza un MonomialDouble ce are ca si coeficient si putere exact aceleasi valori pe care le-ar avea si in cazul anterior.
* Clasa OrderByPut:

Aceasta clasa implementeaza interfata Comparator de tip Monomial pentru a putea ordona descrescator monoamele dintr-un polinom dupa puteri.

* Clasa Polynomial:

1. Ca variabila instanta am un ArrayList de monoame de tip private.

2. Am un getter pentru a returna lista de monoame.

3. Toate metodele pentru efectuarea operatiilor pe polinoame returneaza rezultatul de tip string.

* In metoda **notNull** verific daca am cel putin un monom in lista de monoame
* In metoda **addDuplicates** fac adunarea monoamelor ce au aceeasi putere (in cazul in care utilizatorul introduce monoame ce au aceeasi putere). Se parcurge lista de monoame cu doua for-uri si se verifica constant daca monoamele respective (de pe indexul i si indexul j) au aceeasi putere. Daca da se verifica daca adunarea lor este 0 caz in care le sterg din lista de monoame. Daca adunarea lor nu este 0 atunci se adauga rezultatul la final si dupa se sterg monoamele de pe cei doi indecsi.
* In **constructorul** polinomului se trimite ca parametru un String care este inputul utilizatorului din calculator. Se verifica daca stringul este diferit de ""(daca utilizatorul a introdus ceva). Daca da se sterg toate spatiile cu .replace(" ","");. Se merge dupa pe fiecare string si se inlocuiesc caracterele \*^x cu spatiu si se citesc urmatoarele doua inturi (coeficient si putere). Cu acestea doua se creaza in MonomInt si se adauga in lista de monoame. Daca nu se pot citi atunci se afiseaza un JOptionPane in care se spune sa se introduca un polinom de forma "coeficient\*x^putere + ...". La final se sterg duplicatele apeland metoda addDuplicates dupa care se ordoneaza dupa putere cu OrderByPut.
* In metoda **printResult** se primeste un ArrayList de monoame ce are ca rol sa returneze un string cu monoamele ordonate dupa putere pentru a putea sa fie afisat ca rezultat in GUI.
* In metoda **printResultDivision** se executa acelasi cod ca in metoda printResult, dar monoamele nu mai sunt ordonate. Metoda pentru a putea afisa rezultatul ca fiind cat + rest in urma impartirii polinoamelor.
* In metoda **addPolynomials** se parcurg cu doi indexi cele doua polinoame pentru a putea aduna monoamele cu aceleasi puteri. Pe masura ce se parcurg, in rezultat (ArrayList de Monomial) se adauga monomul cu puterea mai mare si se incrementeaza indexul lui. Daca doua monoame au aceeasi putere atunci se apeleaza metoda de adunare a monoamelor din clasa MonomialInt dupa care se adauga in rezultat suma. Daca se iese din prima bucla (s-a reusit parcurgerea intreaga a unui polinom din cei doi) se verifica care mai are monoame neverificate, caz in care se adauga restul in rezultat.
* In metoda **subPolynomials** este acelasi algoritm ca in addPolynomials numai ca se apeleaza metoda de scadere a doua monoame cand se gasesc doua monoame cu aceeasi putere.
* In metoda **mulPolynomials** se parcurg cele doua polinoame cu doua blucle for each si se aduaga in rezultat inmultirea monoamelor
* In metoda **mulPoMo** are rolul de a returna o lista de monoame ce sunt rezultatul inmultirii unui polinom cu un singur monom (metoda folositoare pentru algoritmul de la impartirea polinoamelor)
* In metoda **divPolunomials** se respecta algoritmul descris la capitolul 3
* In metoda **derPolynomial** se parcurge fiecare monom al polinomului, se deriveaza si se adauga in rezultat
* In metoda **intPolynomial** se parcurge fiecare monom, se integreaza si se adauga la rezultat.
* Clasa Model:

1. Sunt doua variabile instanta de tip private Polynomial ce reprezinta cele doua polinoame introduse de utilizator.

2. Am doua settere pentru a seta cele doua polinoame

3. Metoda checkPolynoms are rolul de a testa daca ambele polinoame au fost introduse, daca niciunul sau doar unul

4. Urmatoarele metode sunt destul de sugestive incat din numele lor ne dam seama de rolul acestora, iar pe deasupra au aceleasi denumiri ca cele din clasa Polynomial

* Clasa View:

1. In constructor se creeaza frame-ul pentru GUI si se apeleaza metodele labels(), textFields() si buttons() pentru a adauga componentele corespunzatoare in frame.

2. Metodele getPol1 si getPol2 au rolul de a returna polinoamele introduse de utilizator sub forma de stringuri (se returneaza textul din textField-uri)

3. Metodele getOp si setOp au rolul de a returna/seta semnul operatiei in labelOp (aflat intre cele doua textField-uri in care se introduc polinoamele) care se doreste a fi efectuata.

4. Metoda setResult seteaza in cel mai de jos textField rezultatul operatiei dorite de utilizator dupa apasarea butonului egal.

5. Urmatoarele sapte metode sunt pentru a seta actionListener pentru fiecare buton din frame.

* Clasa Controller

1. Sunt doua variabile instanta: una de tip Model si cealalta de tip View

2. In constructor se initializeaza cele doua variabile instanta si se adauga cele 7 actionListener care sunt definite ca si clase interne in aceasta clasa.

3. Toate clasele interne (mai putin EqualsListener) au rolul de a seta semnul operatiei in labelOp din clasa View.

In clasa interna EqualsListener se seteaza cele doua polinoame din variabila instanta model si se se ia operatia selectat de user. Se face un switch pe operatie pentru a stii daca ambele polinoame trebuiau introduse (la adunare, scadere, inmultire si impartie) sau doar primul (la derivare si integrare). Dupa acesta se apeleaza metoda corespunzatoare operatiei.

***5) Rezultate***

