



FACULTATEA DE AUTOMATICA SI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

DISCIPLINA TEHNICI DE PROGRAMARE

Documentatie Tema1

- Procesare Polinoame -

Popa Alexandra Maria

Grupa 30224

CTI-ro

Cuprins:

- **Capitolul 1** : Obiectivul Temei..... 3
- **Capitolul 2**: Analiza problemei,modelare,scenarii,cazuri de
utilizare 3
- **Capitolul 3**: Proiectare (decizii de proiectare, diagrame
UML, structuri de date, proiectare clase, interfete, relatii,
packages, algoritmi, interfata utilizator).....5
- **Capitolul 4**: Implementare.....10
- **Capitolul 5**: Rezultate.....13
- **Capitolul 6**: Concluzii.....16
- **Capitolul 7**: Bibliografie.....16

1.Obiectivul Temei:

Enuntul Temei: Propunerea,proiectarea si implementarea unui sistem de procesarea a polinoamelor de o singura variabila cu coeficienti intregi. Operatiile ce vor fi implementate sunt: adunarea, scaderea, inmultirea, impartirea, derivarea si integrarea. De asemenea, proiectul trebuie sa includa o interfata grafica pentru preluarea si afisarea datelor.

Obiectiv principal: Obiectivul acestei teme este proiectarea unui calculator care are capacitatea de a efectua asupra polinoamelor operatii de adunare, scadere, inmultire, impartire, derivare si integreare. Pentru simplificarea interactiunii dintre utilizator si logica din spatele calculatorului se pune la dispozitie o interfata grafica prin intermediul careia utilizarea se face mult mai usor.

Obiective secundare:

- organizarea pe clase
- dezvoltarea algoritmilor folosind POO
- utilizarea structurilor de date
- formularea de scenarii
- implementarea si testarea solutiilor obtinute

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

Problema enuntata necesita atat cunostiinte matematice pentru implementarea operatiilor cerute, cat si cunoasterea programarii orientate pe obiect pentru crearea si manipularea obiectelor.

Din analiza problemei rezulta faptul ca avem nevoie de un set de date de intrare si un set de date de iesire. Datele de intrare sunt reprezentate de doua polinoame in cazul operatiilor de adunare, scadere, inmultire si impartire, iar in cazul derivarii si integrarii este nevoie de un singur polinom.Totodata, operatia in sine reprezinta o data de intrare. Operatia

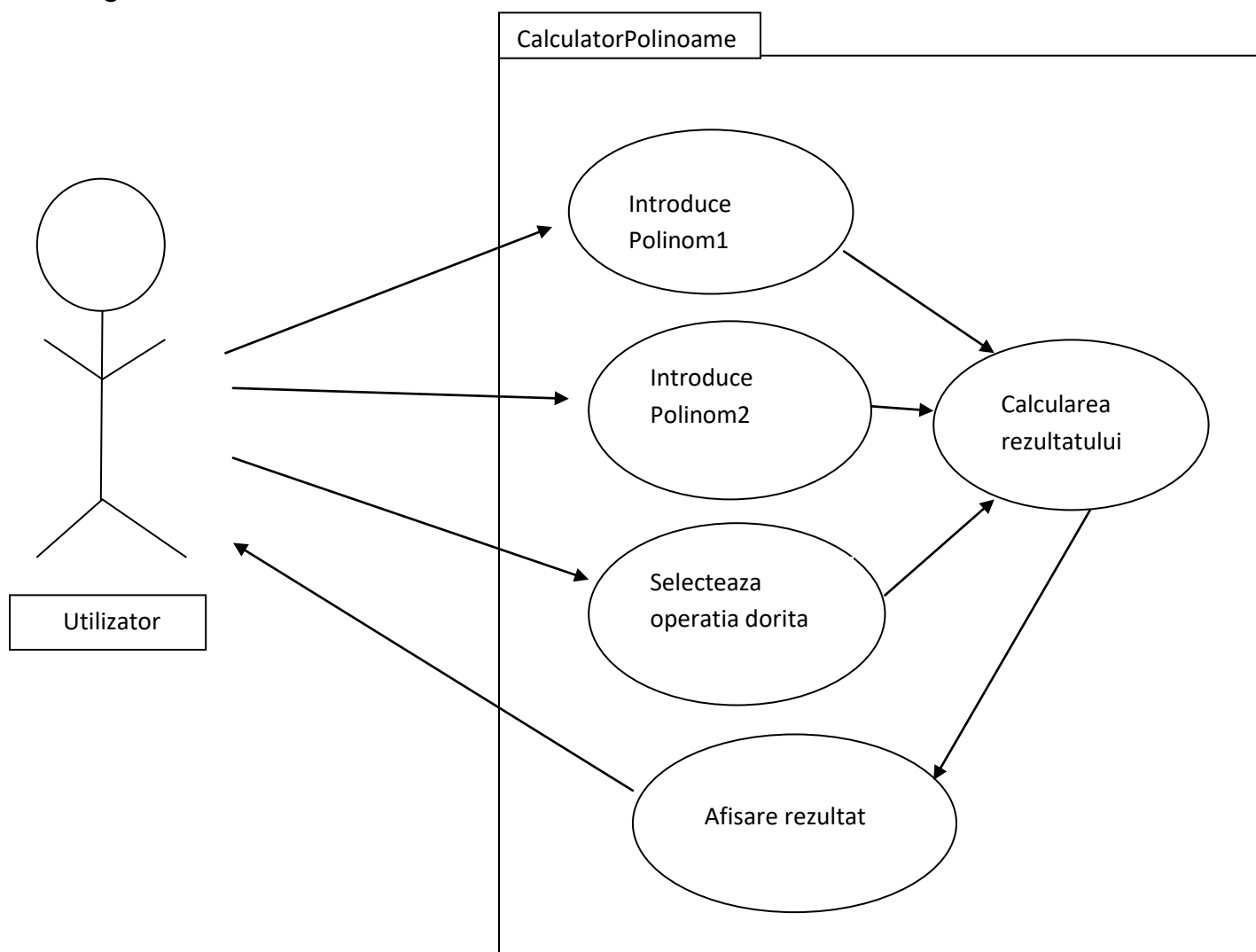
selectata determina generarea datelor de iesire ce sunt reprezentate sub forma de polinom, acesta fiind unul compus in cazul impartirii (cat si rest).

Polinoamele introduse de catre utilizator trebuie sa contina explicit coeficientul intreg si gradul fiecarui monom ce contribuie la formarea polinomului (spre exemplu $5x^3+2x^1+3x^0$). Aceasta metoda de introducere a polinoamelor a fost aleasa special pentru a oferi utilizatorului posibilitatea de a vizualiza fiecare coeficient si fiecare grad in parte. Este necesara introducerea monoamelor ce compun polinomul in ordinea descrescatoare a gradelor. De asemenea, nu este posibila impartirea dintre polinomul 1 si polinomul 2 in cazul in care gradul polinomului 1 este mai mic decat gradul polinomului 2. Orice abatere de la cerintele de introducere a datelor poate duce la ignorarea informatiei introduse si prin urmare se genereaza un rezultat gresit.

Dupa introducerea datelor initiale de catre utilizator, acestea sunt preluate si modelate in functie de operatia ceruta. Datele initiale sunt recunoscute si transformate prin intermediul unui sablon, pentru ca mai apoi sa fie stocate in structuri specifice polinoamelor. Asupra lor se aplica operatii bazate pe algoritmi matematici, iar polinomul rezultat este stocat intr-o structura care poate sa difere in functie de forma rezultatului. De exemplu, in urma operatiei de impartire rezultatul este format din cat si rest.

Pe parcursul procesului de calcul se folosesc reprezentari ale unor polinoame cu coeficienti reali, datorita lucrului cu operatiile de impartire si integrare in urma carora pot rezulta polinoame cu coeficienti reali. Rezultatul obtinut va fi pus la dispozitia utilizatorului intr-o forma explicita compusa din polinoame cu coeficienti reali doar in cazul operatiilor de impartire sau integrare, iar in rest coeficientii vor fi afisati sub forma unor numere intregi . Afisarea trebuie sa fie clara pentru utilizator, scopul fiind ca produsul final sa fi cat mai practic si usor de folosit.

Diagrama use-case:



3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator) :

Proiectarea procesorului de polinoame a necesitat crearea a 13 clase: *Monom*, *Polinom*, *AddOp*, *SubOp*, *MulOp*, *DivOp*, *DerivOp*, *IntegOp*, *Result*, *Bipolinom*, *ViewCalculator*, *ControllerCalculator*, *CalculatorPolinoame* ; precum si crearea interfetei *IOperation* care este implementata de clasele specifice fiecarei operatii mentionate mai

sus. Interfața grafică se bazează pe șablonul MVC ce presupune separarea proiectului în Model, View și Controller, în funcție de nevoi, reliefând astfel principiile POO. În continuare voi prezenta proiectarea în mare a claselor ce formează aplicația, metodele acestora fiind descrise mai pe larg în capitolul 4.

Monom -> această clasă are două variabile instanță: *coef*- coeficientul monomului, de tip float și *degree*- gradul monomului, de tip int. Coeficientul este de tip float, datorită faptului că se utilizează operații de împărțire și integrare, ceea ce presupune calculul cu numere reale. Clasa conține metode de tip setter și getter: *getCoef()*, *getDegree()*, *setCoef(float coef)*, *setDegree(int degree)* și implementează metodele *toString()* și *compareTo(Monom m)*.

Polinom -> conține o singură variabilă instanță - *elem*, de tipul *List<Monom>*, deoarece se dorește stocarea unui număr variabil de monoame, specific pentru fiecare polinom. Clasa are două tipuri de constructor și implementează următoarele metode: *getElem()*, *setElem(List<Monom> elem)*, *addMonom(Monom m)* și *toString()*;

IOperation -> reprezintă o interfață ce conține metoda *compute(Polinom p1, Polinom p2)*.

AddOp, *SubOp*, *MulOp*, *DivOp*, *DerivOp*, *DerivOp* -> clase ce implementează interfața *IOperation()*, fiecare având o delegare specializată. Algoritmii folosiți de fiecare dintre aceste clase în implementarea metodei *compute(Polinom p1, Polinom p2)* sunt prezentați mai pe larg în capitolul 4.

Result -> conține o singură variabilă instanță- *rezultat*, de tipul *Polinom*, utilizată în lucrul cu valorile obținute în urma apelării metodei *Compute(...)*; are două tipuri de constructor, un getter și un setter: *getRezultat()* și *setRezultat(Polinom rezultat)*.

Bipolinom -> această clasă extinde clasa *Result()* și conține o variabilă instanță -*rest*, de tip *Polinom*, folosită special pentru rezultatele operațiilor

din care rezulta atat cat si rest; contine un constructor si o metoda de get si set: *getRest()* si *setRest(Polinom rest)*.

Clasele prezentate mai sus reprezinta modelul aplicatiei , deoarece acestea definesc logica din spatele procesarii datelor de intrare.

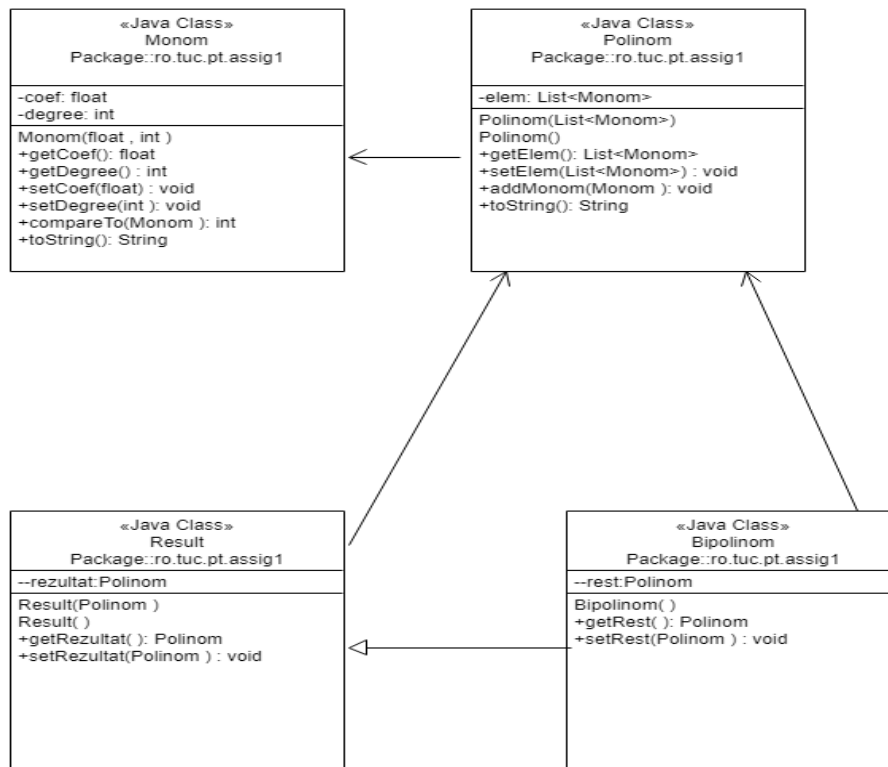
ViewCalculator -> reprezinta interfata grafica a aplicatiei; prin intermediul elementelor care o compun se creaza legatura cu utilizatorul, usurand astfel considerabil modul de utilizare a procesorului de polinoame . Pentru realizarea interfetei grafice am folosit 15 variabile instanta, dintre care: 4 campuri de tipul *TextField*, 2 editabile pentru introducerea polinoamelor (*firstPolinom*, *secondPolinom*) si 2 pentru afisarea rezultatului (cat si rest) care nu pot fi editate de catre utilizator, ci doar vizualizate (*calcRezultat*, *calcRest*); 5 etichete de tip *JLabel* care indica semnificatia campurilor text sau a butoanelor (*labelP1*, *label P2*, *labelRezultat*, *labelRest*, *labelOperatii*); 6 butoane de tipul *Button*, unul pentru fiecare tip de operatie in parte (*addBtn*, *subBtn*, *mulBtn*, *divBtn*, *derivBtn*, *integBtn*). Constructorul clasei contine declararea a 5 panel-uri de tip *JPanel*, in care au fost adaugate variabilele instanta ce compun fereastra de vizualizare. Acesta sunt organizate in asa fel incat sa se pastreze o anumita aliniere a elementelor in fereastra in functie de axele Ox si Oy. Totodata in acest constructor se seteaza vizibilitatea ferestrei principale si numele acesteia. Clasa contine de asemenea si metode de tip getter si setter care preiau informatia introdusa de catre utilizator in campurile *firstPolinom* si *secondPolinom* si o transmit mai departe, si seteaza rezultatul obtinut in urma procesarii datelor de intrare (*getFirstPolinom()*, *getSecondPolinom()*, *setCalcResult (String addCalcRezultat) setCalcRest(String addCalcRest())*). Pe langa aceste, se gasesc si metode care adauga ascultatori (action listeners) pentru fiecare buton in parte, precum: *addAddBtnListener(ActionListener l)*, *addSubBtnListener(ActionListener l)*, *addMulBtnListener(ActionListener l)*, *addDivBtnListener(ActionListener l)*, *addDerivBtnListener(ActionListener l)*, *addIntegBtnListener(ActionListener l)* .

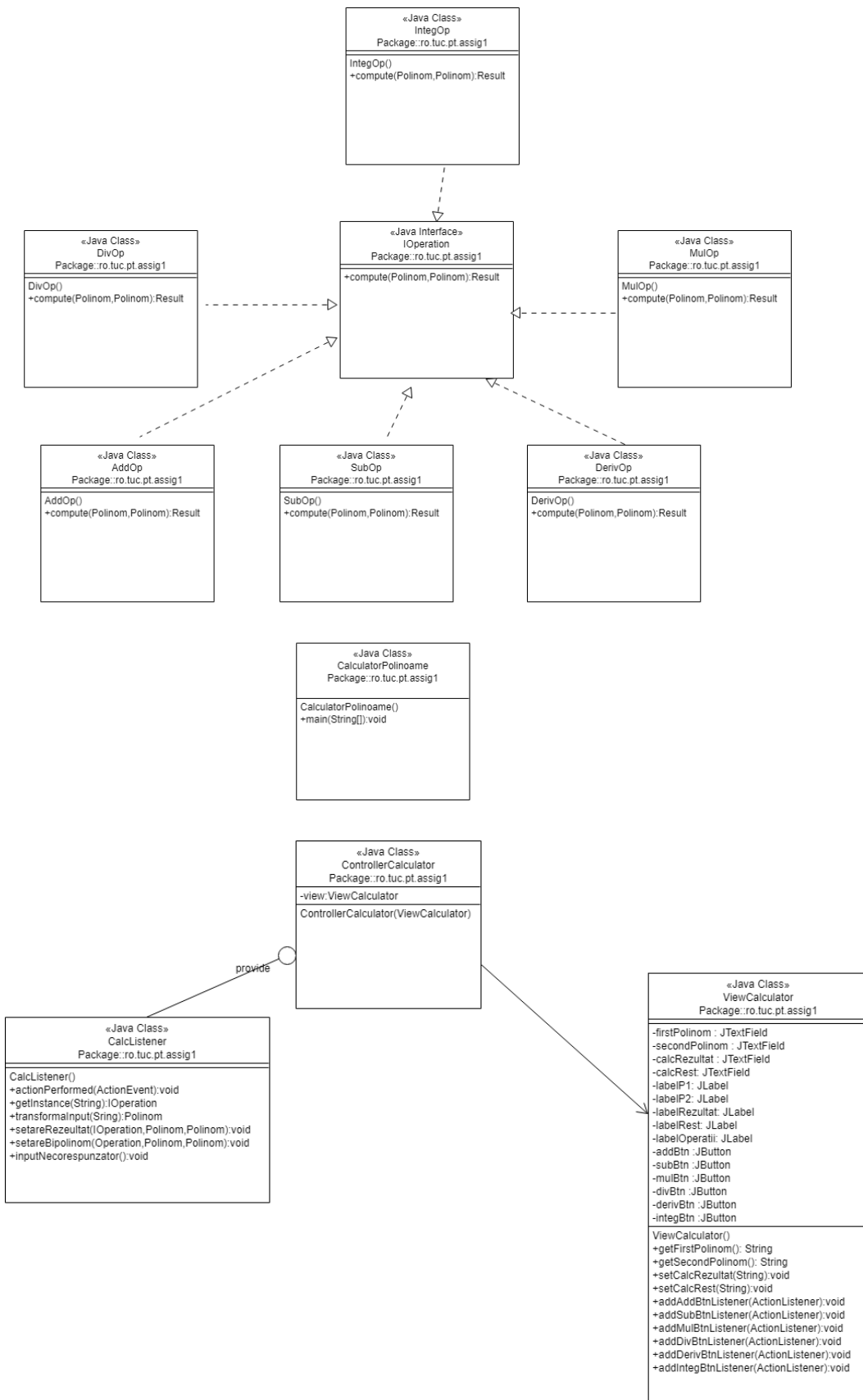
ControllerCalculator -> reprezinta partea de control a proiectului, ce decide ce pasi urmeaza sa faca modelul. Clasa contine o singura variabila instantata- *view*, de tipul *ViewCalculator*, ce realizeaza legatura cu clasa

ViewCalculator. Constructorul clasei contine ascultatori pentru fiecare din butoane, care sunt adaugati la view. Clasa de control contine o clasa interna *CalcListener* ce implementeaza *ActionListener* si contine metoda *actionPerformed(ActionEvent e)*, *getInstance(String s)*, *transformaInput(String s)*, *setareRezultat(IOperation op, Polinom p1, Polinom p2)*, *setareBipolinom(IOperation op, Polinom p1, Polinom p2)*, *inputNecorespunzator()* .

CalculatorPolinoame -> contine programul principal care initializeaza interfata si leaga componentele impreuna.

In continuare este prezentata diagrama UML de clase a proiectului:





4. Implementare :

In acest capitol se vor descrie deciziile de implementare ale metodelor fiecărei clase în parte, împreună cu algoritmi ce sunt folosiți în realizarea acestora .

Monom :

- *compareTo()* -> metoda implementată cu scopul de a se putea realiza sortarea monoamelor în funcție de gradul acestora.
- *toString()* -> realizează o transformare a monomului în String; metoda folosită pentru afișare; returnează un String.

Polinom :

- *addMonom(Monom m)* -> adaugă un nou monom în lista de monoame din care este format polinomul.
- *toString()* -> metoda folosită pentru afișarea polinomului sub formă de String; în interiorul acesteia se apelează metoda *toString()* din clasa *Monom*; returnează un String.

AddOp:

- *compute(Polinom p1, Polinom p2)* -> returnează un obiect de tip Result. Această metodă realizează operația de adunare a polinoamelor p1 și p2 astfel: se setează o variabilă pe post de semafor. Se parcurge p1 și p2 și se compară gradul fiecărui monom din p1 cu gradul fiecărui monom din p2. În caz de egalitate se adună coeficienții monoamelor și suma rezultată devine coeficientul noului monom creat care se adaugă polinomului rezultat, doar dacă suma este diferită de 0. Totodată se schimbă variabila semafor. Dacă la finalul parcurgerii monoamelor din p2 semaforul nu s-a schimbat, înseamnă că nu s-a găsit nici un monom cu același grad ca al monomului verificat, deci trebuie adăugat la polinomul rezultat. În continuare se compară gradul fiecărui monom din p2 cu gradul fiecărui monom din p1. Este nevoie de a doua parcurgere deoarece în acest mod se evită pierderea monomamelor ce compun p2 și au un grad diferit de gradele monomelor ce se găsesc în p1. De

aceea, in caz de egalitate nu se va mai adauga un nou polinom la rezultat (pentru a evita duplicarea) ci doar se va schimba semaforul pentru a putea verifica daca vreun monom din p2 este adaugat la rezultaul final. Daca nu este in rezultat se adauga ulterior.

SubOp:

- *compute(Polinom p1, Polinom p2)* -> returneaza un obiect de tip Result. Aceasta metoda realizeaza operatia de scadere a polinoamelor p1 si p2 astfel: se vor parcurge toate monoamele ce compun p2 si coeficientii acestora se vor inmulti cu -1. Apoi se realizeaza operatia de adunare intre p1 si noul p2.

MulOp:

- *compute(Polinom p1, Polinom p2)* -> returneaza un obiect de tip Result. Aceasta metoda realizeaza operatia de inmultire a polinoamelor p1 si p2 astfel: se parcurg cele doua polinoame si se realizeaza inmultirea coeficientului fiecarui monom din p1 cu coeficientul fiecarui monom din p2 si se aduna gradele acestora. Dupa ce fiecare monom din p1 a fost inmultit cu p2 se realizeaza adunarea rezultatului la rezultatele obtinute precedent pentru a evita ca rezultatul final sa contina monoame cu grade egale.

DivOp :

- *compute(Polinom p1, Polinom p2)* -> returneaza un obiect de tip Bipolinom. Aceasta metoda realizeaza operatia de impartire a polinomului p1 la polinomul p2 astfel: intai se verifica daca coeficientul primului polinom este 0. In acest caz, rezultatul va fi 0. In caz contrar se apeleaza la o structura repetitive se realizeaza cat timp gradul lui p1 este mai mare sau egal decat gradul lui p2. Se creaza un nou monom a carui coeficient este rezultatul obtinut in urma impartirii coeficientul primului monom din p1 la primul monom din p2, iar gradul este rezultatul scaderii coeficientilor acestor 2 monoame. Noul monom se adauga unui polinom care mai apoi se

inmulteste cu p2, iar rezultatul se scade din p1. Se seteaza noul cat si noul rest .

DerivOp :

- *compute(Polinom p1, Polinom p2)* -> returneaza un obiect de tip Result. Aceasta metoda realizeaza operatia de derivare a polinomului p1 astfel : daca gradul polinomului este diferit de 0, atunci se va parcurge p1 monom cu monom si se va inmulti fiecare coeficient cu gradul corespunzator ,iar gradul va scadea cu 1.

IntegOp:

- *compute(Polinom p1, Polinom p2)* -> returneaza un obiect de tip Result. Aceasta metoda realizeaza operatia de integrare a polinomului p1 astfel : se parcurge p1 si se imparte coeficientul fiecarui monom la gradul corespunzator + 1, iar gradul se va aduna cu 1.

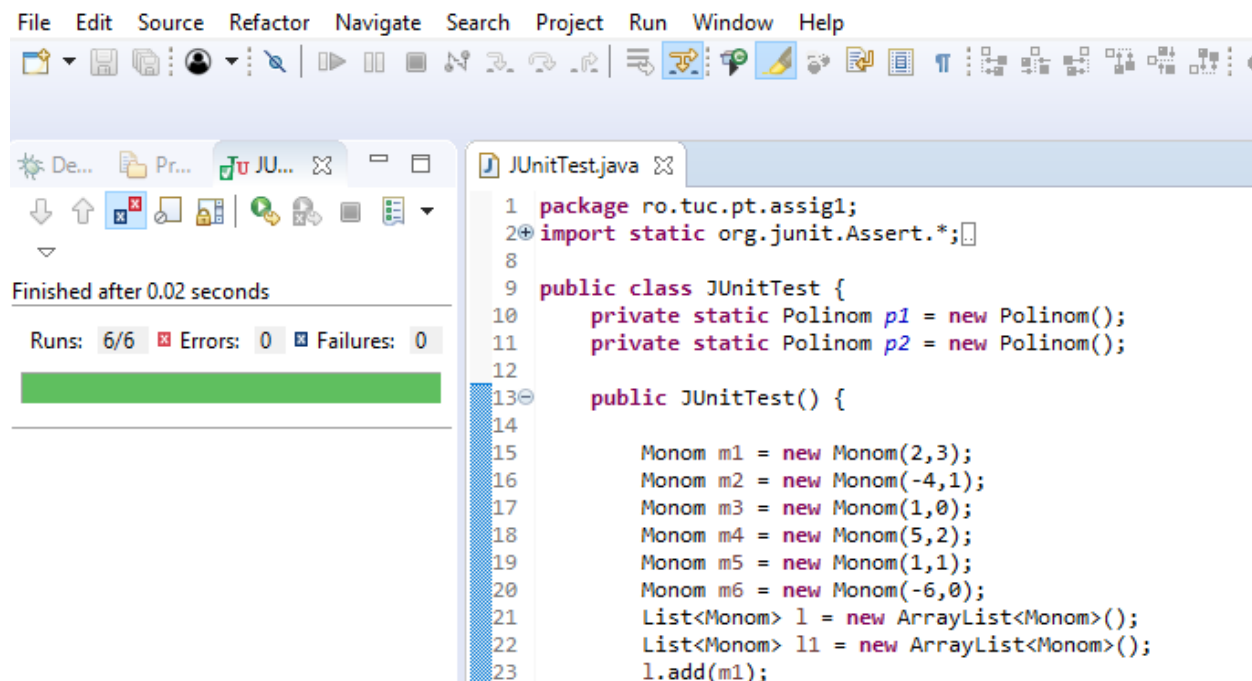
ControllerCalculator :

- *actionPerformed(ActionEvent e)* => aceasta metoda realizeaza comunicarea cu clasa *ViewCalculator()*; se preiau datele de intrare sub forma de string si se transforma in obiecte de tipul Polinom prin apelul metodei *transformaInput(String s)*;se obtine sub forma de string denumirea butonului apasat si se transforma intr-un obiect de tip IOperation prin intermediul metodei *getInstance(String s)*. Se verifica operatia selectata de catre utilizator. Daca aceasta este o impartire se verifica impartitorul care trebuie sa fie diferit de 0 si gradul primului polinom sa fie mai mare sau egal decat gradul celui de al doilea. In cazul in care inputul nu respecta conditiile se va semnala acest fapt , iar in caz contrar se va afisa catul si retul impartirii . In cazul in care operatia este oricare alta inafara de impartire, se afiseaza direct rezultatul, campul fiind gol.
- *getInstance(String s)* -> primeste operatia sub forma de String si o transforma in IOperation.

- *transformInput(String s)* -> transforma stringul introdus de utilizator in Polinom.
- *setareRezultat(IOperation op, Polinom p1, Polinom p2)* -> seteaza campurile pentru rezultat cu valoarea corespunzatoare rezultatului obtinut in model, iar campul pentru rest devine null.
- *setareBipolinom(IOperation op, Polinom p1, Polinom p2)* -> seteaza campurile pentru cat si rest cu valorile corespunzatoare rezultatului obtinut in model.
- *inputNecorespunzator()* -> seteaza campurile pentru cat si rest pe null

4. Rezultate :

Testarea proiectului se face cu ajutorul clasei JUnit, ce foloseste metode de test pentru fiecare dintre operatii : *testAdd()*, *testSub()*, *testMult()*, *testDiv()*, *testDeriv()*, *testInteg()*. De asemenea s-a utilizat metoda *assertTrue(boolean condition)* care verifica daca conditia pusa este adevarata si ca urmare daca testul a trecut.



The screenshot shows an IDE with the following components:

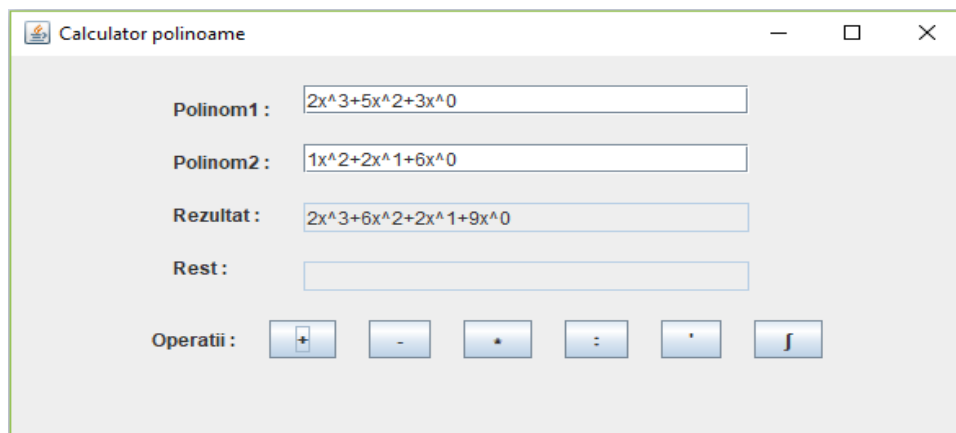
- Top Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard IDE icons for file operations, running, and debugging.
- Left Panel (Project Explorer):** Shows a project structure with folders like 'De...' and 'Pr...'. Below it, a status bar indicates 'Finished after 0.02 seconds' and 'Runs: 6/6', 'Errors: 0', 'Failures: 0'.
- Right Panel (Editor):** Displays the code for `JUnitTest.java`. The code is as follows:


```

1 package ro.tuc.pt.assig1;
2 import static org.junit.Assert.*;
3
4
5
6
7
8
9 public class JUnitTest {
10     private static Polinom p1 = new Polinom();
11     private static Polinom p2 = new Polinom();
12
13     public JUnitTest() {
14
15         Monom m1 = new Monom(2,3);
16         Monom m2 = new Monom(-4,1);
17         Monom m3 = new Monom(1,0);
18         Monom m4 = new Monom(5,2);
19         Monom m5 = new Monom(1,1);
20         Monom m6 = new Monom(-6,0);
21         List<Monom> l = new ArrayList<Monom>();
22         List<Monom> l1 = new ArrayList<Monom>();
23         l.add(m1);
      
```

5. Rezultate:

Adunare :



Calculator polinoame

Polinom1 : $2x^3+5x^2+3x^0$

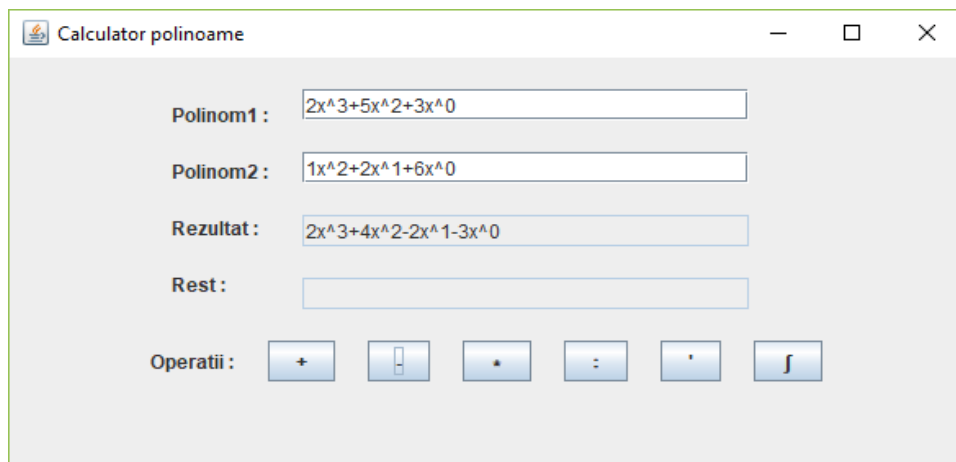
Polinom2 : $1x^2+2x^1+6x^0$

Rezultat : $2x^3+6x^2+2x^1+9x^0$

Rest :

Operatii : $+$ $-$ $*$ $:$ $^$ f

Scadere :



Calculator polinoame

Polinom1 : $2x^3+5x^2+3x^0$

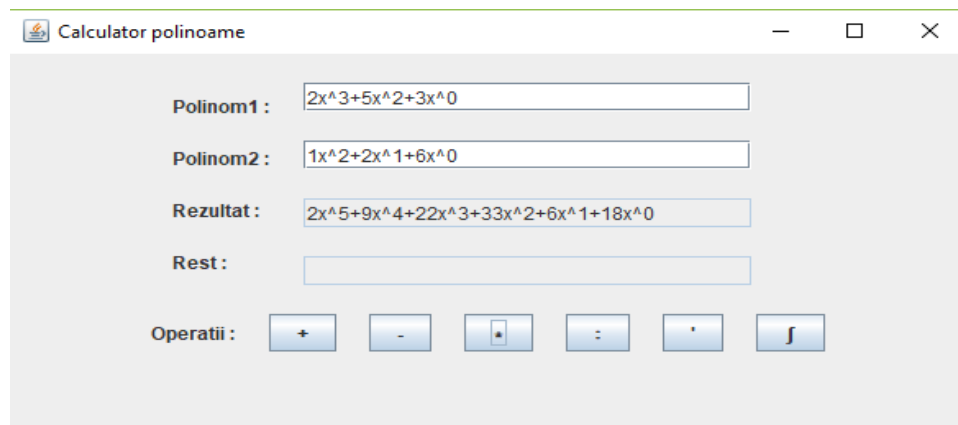
Polinom2 : $1x^2+2x^1+6x^0$

Rezultat : $2x^3+4x^2-2x^1-3x^0$

Rest :

Operatii : $+$ $-$ $*$ $:$ $^$ f

Inmultire :



Calculator polinoame

Polinom1 : $2x^3+5x^2+3x^0$

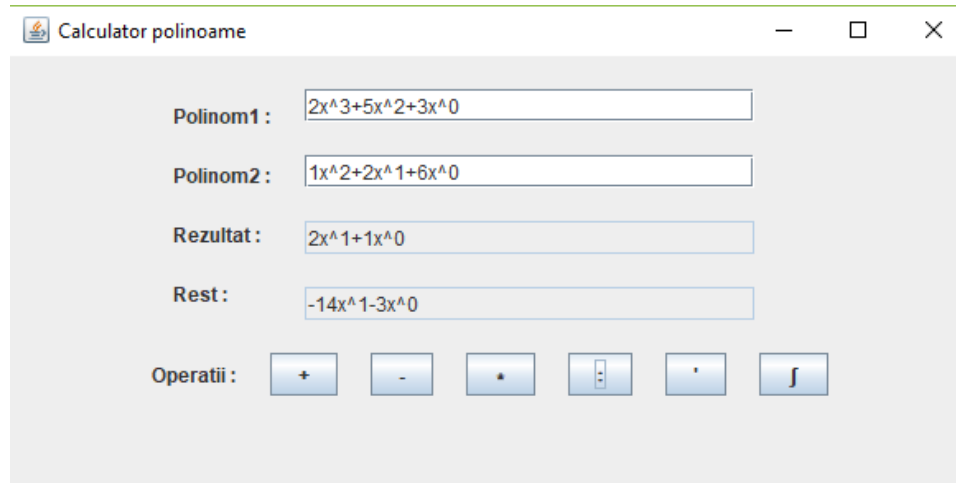
Polinom2 : $1x^2+2x^1+6x^0$

Rezultat : $2x^5+9x^4+22x^3+33x^2+6x^1+18x^0$

Rest :

Operatii : $+$ $-$ $*$ $:$ $^$ f

Impartire :



Calculator polinoame

Polinom1 : $2x^3+5x^2+3x^0$

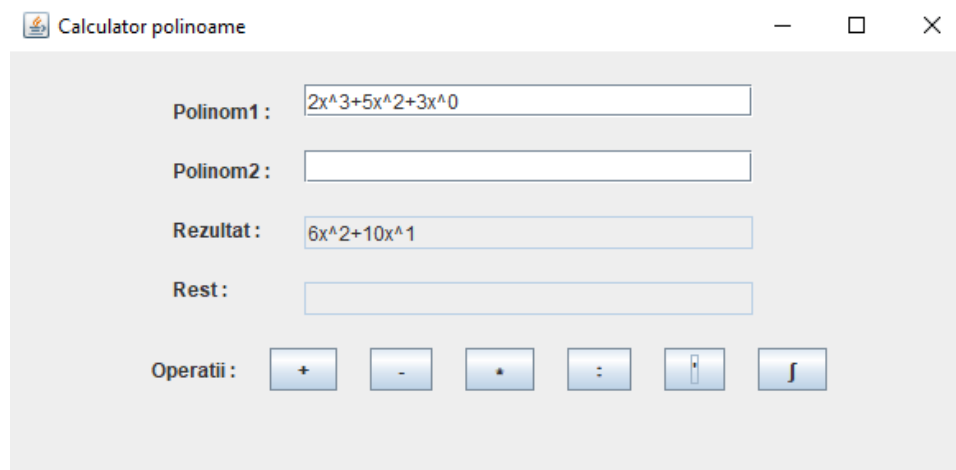
Polinom2 : $1x^2+2x^1+6x^0$

Rezultat : $2x^1+1x^0$

Rest : $-14x^1-3x^0$

Operatii : $+$ $-$ $*$ $:$ $'$ \int

Derivare :



Calculator polinoame

Polinom1 : $2x^3+5x^2+3x^0$

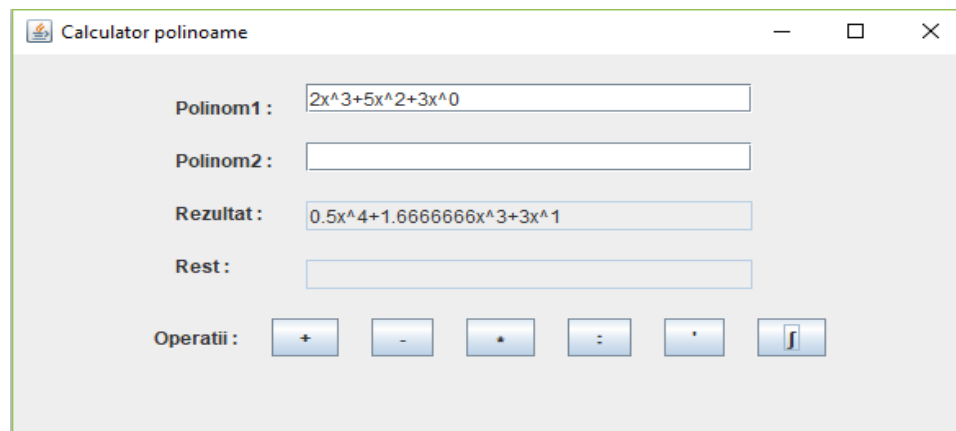
Polinom2 :

Rezultat : $6x^2+10x^1$

Rest :

Operatii : $+$ $-$ $*$ $:$ $'$ \int

Integrare :



Calculator polinoame

Polinom1 : $2x^3+5x^2+3x^0$

Polinom2 :

Rezultat : $0.5x^4+1.6666666x^3+3x^1$

Rest :

Operatii : $+$ $-$ $*$ $:$ $'$ \int

6. Concluzii :

În urma acestei teme am învățat să îmi structurez mai bine codul astfel încât acesta să respecte principiile POO. Am fost pusă în situația de a realiza o interfață grafică, ceea ce a adus un plus considerabil cunoștințelor legate de lucrul cu modelul MVC, cât și în ceea ce privește modulul de a scrie cod.

Ca îmbunătățiri ulterioare, aplicația mea ar putea beneficia de adăugarea unor noi operații care să poată fi realizate. Totodată, calculatorul ar putea avea obținerea de calculator standard, programmer sau științific, iar fiecare tip în parte să conțină operații specifice.

7. Bibliografie :

- Indrumator de laborator POO
- Curs POO
- <https://stackoverflow.com/>