

**FACULTATEA DE AUTOMATICA SI CALCULATOARE**

**DEPARTAMENTUL CALCULATOARE**

**DISCIPLINA TEHNICI DE PROGRAMARE**

**Documentatie Tema 1**

**Sistem de procesare a polinoamelor**

Pavel Madalina Adriana

Grupa 30224

**CUPRINS**

**1. OBIECTIVUL TEMEI ……………………………………………………………………..3**

**2. ANALIZA PROBLEMEI, MODELARE, SCENARII, CAZURI DE UTILIZARE …...4**

**3. PROIECTARE (DECIZII DE PROIECTARE, DIAGRAME UML, STRUCTURI DE DATE, PROIECTARE CLASE, INTERFETE, RELATII, PACKAGES, ALGORITMI, INTERFATA UTILIZATOR) …………………………..……………………………………7**

**4. IMPLEMENTARE ………………………………………………………………………..10**

**5. REZULTATE …………………………………….………………………………………..12**

**6. CONCLUZII .……………………………………………………………………………....12**

**7. BIBLIOGRAFIE …………………………………………………………………………..13**

**1.OBIECTIVUL TEMEI**

Enuntul temei este urmatorul :

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

Obiectivul principal al acestei teme este crearea unui calculator care sa permita realizarea operatiilor de baza asupra polinoamelor (adunare, scadere, inmultire, impartire, derivare si integrare); un calculator precum cel din Windows, care efectueaza operatii pe numere, doar ca de data aceasta operatiile se adreseaza polinoamelor. Cerintele impuse sunt : implementarea operatiilor de adunare, scadere, inmultire, impartire, derivare si integrare, la care se adauga si necesitatea unui design OOP, iar efectuarea testelor sa se realizeze prin JUnit.

Obiectivele secundare sunt urmatoarele :

* Dezvoltarea de use-case-uri si scenarii (Capitolul 2)
* Alegerea structurilor de date (Capitolul 3)
* Impartirea pe clase, utilizarea interfetelor, evidentierea relatiilor (Capitolul 3)
* Dezvoltarea algoritmilor utilizati (Capitolul 4)
* Testarea solutiei (Capitolul 5)

**2.ANALIZA PROBLEMEI, MODELARE, SCENARII, CAZURI DE UTILIZARE**

Enuntul problemei specifica faptul ca polinoamele asupra carora se efectueaza operatiile sunt cu coeficienti intregi, aspect ce conduce la aparitia unei probleme in cazul operatiilor de impartire si integrare. Operatia de impartire presupune printre altele si impartirea coeficientilor intregi, al caror cat va fi reprezentat de un numar real. Aceeasi problema apare si in cazul integrarii, cand coeficientul se va imparti la puterea monomului (adunata cu 1), care este tot un numar intreg, rezultatul acestei impartiri fiind un numar real. Din acest motiv, am ales sa declar coeficientul monomului ca fiind o variabila de tipul float, pentru a rezolva problema impartirii si a integrarii.

In ceea ce priveste modelarea problemei, am incercat sa raspund urmatoarelor intrebari : Cum sunt introduse datele de intrare si care sunt acestea? Cum vor fi salvate aceste date de intrare si care este structura potrivita ce trebuie utilizata pentru a usura modul de procesare al acestor date?

Datele de intrare sunt reprezentate de cele 2 polinoame (in cazul operatiilor de adunare, scadere, inmultire, impartire) sau doar de un polinom (in cazul operatiilor de derivare si integrare). Prin intermediul interfetei GUI puse la dispozitie de aplicatie, cele 2 polinoame sunt introduse de utilizator, respectand formatul din exemplul urmator : 1x^2-2x^1+3x^0. Acestea vor fi salvate in obiecte de tipul String. Cu ajutorul unui regex, aplicatia va extrage din cele 2 String-uri informatiile necesare pentru efectuarea operatiilor, si anume lista de coeficienti si puteri care caracterizeaza monoamele din care sunt alcatuite polinoamele. Asadar, vom privi polinoamele ca o lista de monoame, care la randul lor reprezinta perechi formate dintr-un coeficient si o putere, si vom efectua operatiile cerute pe aceste liste.

In continuare se va prezenta un scenariu pentru operatia de adunare. Actorii acestui scenariu sunt utilizatorul care doreste sa efectueze operatia de adunare si calculatorul de polinoame reprezentat de aplicatie. Utilizatorul trebuie sa introduca polinoamele pe care doreste sa le adune in campurile denumite “Polinom 1” si “Polinom 2”, avand grija sa respecte formatul impus, precum in exemplul : “1x^2-2x^1+3x^0”. In cazul in care polinoamele au fost introduse in formatul corect, dupa ce utilizatorul apasa butonul corespunzator operatiei dorite, in acest caz butonul marcat cu “+”, aplicatia va citi cele 2 polinoame introduse si, prin intermediul unui regex, va selecta informatiile de interes pentru efectuarea operatiei : listele de coeficienti si puteri care definesc polinomul. Aplicatia va realiza adunarea, rezultatul fiind afisat in campul denumit “Rezultat”, in timp ce campul “Rest” va ramane gol deoarece acesta este utilizat doar in cadrul operatiei de impartire, pentru a afisa restul impartirii.

In cazul in care utilizatorul a introdus din greseala un monom cu o putere negativa sau un coeficient real sau un polinom intr-un format diferit de cel prezentat mai sus, toti termenii care nu respecta formatul impus vor fi ignorati, iar aplicatia va efectua operatia doar pe acei termeni care au fost introdusi corect.

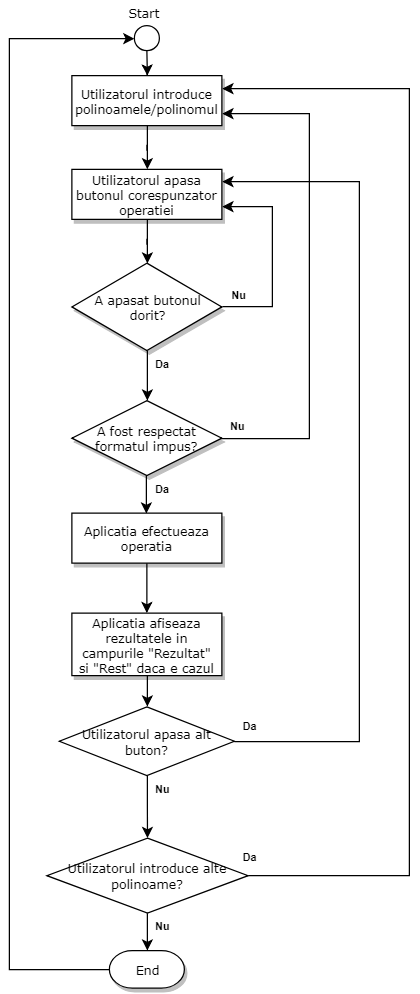
In cazul in care se introduce doar un polinom si se apasa butonul “+”, aplicatia va afisa exact polinomul introdus de utilizator, interpretand acest lucru ca fiind o adunare cu un monom cu coeficientul si puterea 0.

Polinoamele se introduc in aceasta ordine : prima data se completeaza campul “Polinom 1”, apoi campul “Polinom 2”. In cazul in care nu se introduce niciun polinom si se apasa oricare din butoanele care desemneaza operatiile, aplicatia va afisa mesajul : “Introduceti cel putin un polinom!”. Acest mesaj va fi afisat si in cazul in care s-a introdus un singur polinom, doar in campul “Polinom 2”.

Un alt scenariu pe care il putem lua in considerare este cel pentru operatia de impartire. Actorii raman aceiasi, datele de intrare se introduc in modul prezentat mai sus, singura diferenta constand in afisarea rezultatului. Impartirea a doua polinoame poate genera aparitia unui rest, pe langa cat. In acest caz, rezultatul impartirii va aparea in cele 2 campuri puse la dispozitie pentru afisarea rezultatelor : “Rezultat” (pentru cat) si “Rest” (pentru rest). In cazul in care restul impartirii este 0, in campul “Rest” nu se va afisa nimic.

In cazul in care utilizatorul introduce din greseala un singur polinom (doar in campul “Polinom 1”), aplicatia nu va transmite niciun rezultat deoarece asteapta introducerea ambelor polinoame. In schimb, in cazul in care utilizatorul introduce ambele polinoame, iar in campul “Polinom 2” introduce un polinom avand toti coeficientii 0, aplicatia va afisa mesajul : “Ati incercat sa efectuati o impartire la 0!”, deoarece acest lucru nu este permis.

Un ultim scenariu este cel corespunzator operatiilor de derivare si integrare. In cadrul acestor operatii este nevoie doar de un singur polinom, pe care utilizatorul trebuie sa il introduca in campul “Polinom 1”, deoarece doar acest camp va fi luat in considerare, rezultatul fiind afisat in campul “Rezultat”. In cazul in care utilizatorul introduce un polinom si in campul “Polinom 2”, acest polinom nu va influenta cu nimic rezultatul operatiei.



UML Activity Diagram

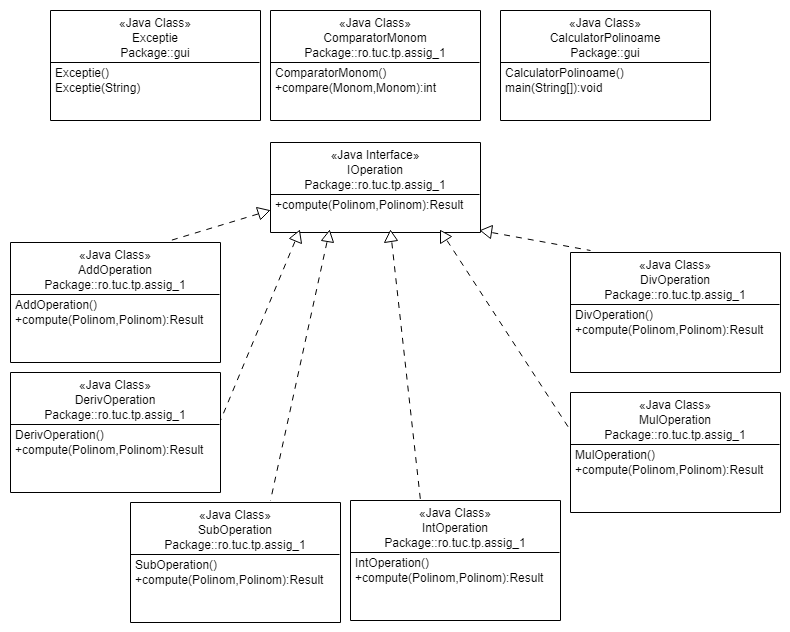
**3. PROIECTARE (DECIZII DE PROIECTARE, DIAGRAME UML, STRUCTURI DE DATE, PROIECTARE CLASE, INTERFETE, RELATII, PACKAGES, ALGORITMI, INTERFATA UTILIZATOR)**

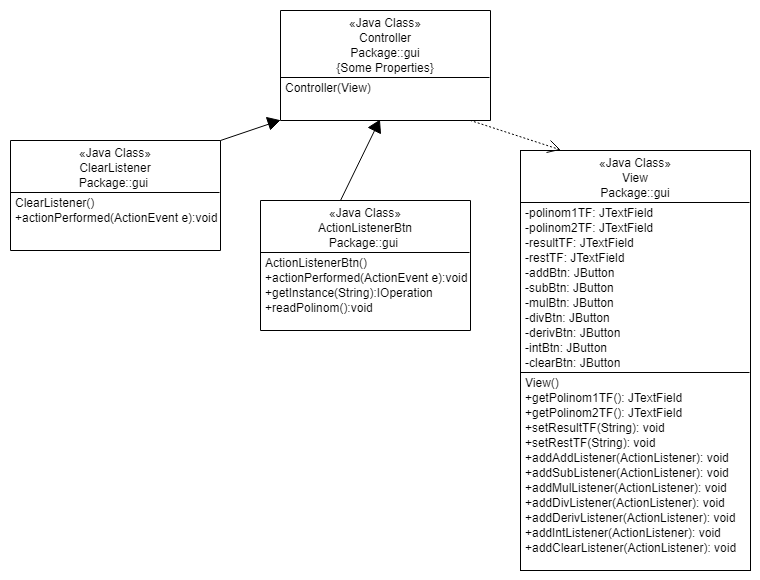
Pentru implementarea aceste probleme am utilizat urmatoarele clase si interfete :

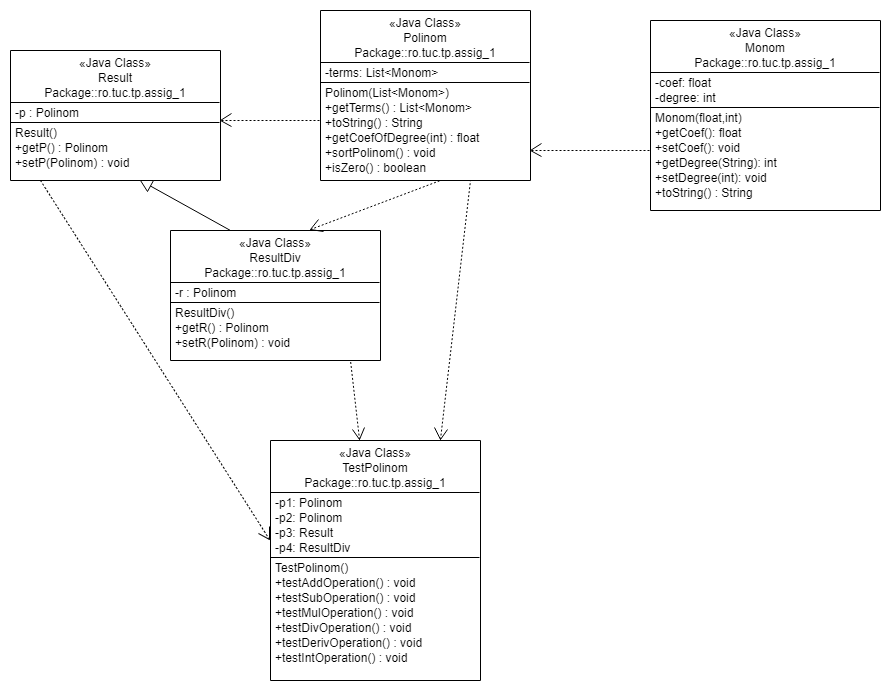
1. **Clasa Monom** – contine 2 variabile instanta, care descriu un monom : coeficientul (de tip float) si gradul (de tip int). Clasa are un constructor, iar pe langa constructor exista settere si gettere pentru ambele variabile instanta si metoda toString.
2. **Clasa Polinom** – contine o singura variabila instanta : o lista de monoame de tipul List<Monom>, ce reprezinta termenii polinomului. Clasa contine un constructor, un getter, metoda toString si alte 3 metode (o metoda care returneaza coeficientul monomul al carui grad este dat, o metoda care sorteaza un polinom si o metoda care verifica daca polinomul are toti coeficientii 0).
3. **Clasa ComparatorMonom** – implementeaza interfata Comparator<Monom> si contine doar metoda compare, care stabileste modul de comparare a doua monoame.
4. **Clasa Result** – contine o variabila instanta de tip Polinom, in care se va stoca rezultatul operatiei (sau catul in cazul operatiei de impartire). Pe langa constructor, clasa contine un setter si un getter.
5. **Clasa ResultDiv** – extinde clasa Result, are in plus o variabila instanta de tipul Polinom, utilizata pentru a stoca restul rezultat in urma operatiei de impartire. Si aceasta clasa contine doar un constructor, un setter si un getter.
6. **Interfata IOperation** – declara sablonul metodei compute, metoda ce va fi implementata de clasele care extind aceasta interfata.
7. **Clasa AddOperation** – implementeaza interfata IOperation, descrie corpul metodei compute, care in acest caz realizeaza operatia de adunare a 2 polinoame.
8. **Clasa SubOperation** – implementeaza interfata IOperation, descrie corpul metodei compute, care in acest caz realizeaza operatia de scadere a 2 polinoame.
9. **Clasa MulOperation** – implementeaza interfata IOperation, descrie corpul metodei compute, care in acest caz realizeaza operatia de inmultire a 2 polinoame.
10. **Clasa DivOperation** – implementeaza interfata IOperation, descrie corpul metodei compute, care in acest caz realizeaza operatia de impartire a 2 polinoame.
11. **Clasa DerivOperation** – implementeaza interfata IOperation, descrie corpul metodei compute, care in acest caz realizeaza operatia de derivare a unui polinom.
12. **Clasa IntOperation** – implementeaza interfata IOperation, descrie corpul metodei compute, care in acest caz realizeaza operatia de integrare a unui polinom.
13. **Clasa TestPolinom** – contine 4 variabile instanta statice : 2 de tipul Polinom, una de tipul Result si una de tipul ResultDiv. Aceasta clasa este utilizata pentru testarea operatiilor cu ajutorul JUnit.
14. **Clasa CalculatorPolinoame** – nu are variabile instanta, contine doar metoda main in care se instantiaza un obiect de tipul View si un obiect de tipul Controller. Lansarea in executie a metodei main din aceasta clasa deschide calculatorul (aplicatia).
15. **Clasa Controller** – are ca variabila instanta un obiect de tipul View; contine un constructor in care se adauga acelasi listener fiecarui buton in parte, cu exceptia butonului de clear (“C”). Aceasta clasa contine alte 2 clase interne numite **ActionListenerBtn** si **ClearListener**, care implementeaza ActionListener si descriu comportamentele butoanelor.
16. **Clasa Exceptie** – extinde clasa Exception, nu are variabile instanta si contine 2 constructori. Este utilizata pentru aruncarea unei exceptii in momentul in care utilizatorul apasa un buton fara a introduce un polinom macar in campul “Polinom 1” (campul “Polinom 2” este optional deoarece nu toate operatiile necesita 2 polinoame).
17. **Clasa View** – extinde clasa JFrame, are ca variabile instanta 4 obiecte de tipul JTextField, utilizate pentru campurile de text, si 7 obiecte de tipul JButton, care reprezinta butoanele. Clasa are un constructor in care se realizeaza aranjarea componentelor in fereastra, deci interfata grafica a calculatorului. Pe langa acest constructor, exista si gettere pentru campurile in care se introduc polinoamele si settere pentru campurile folosite la afisarea rezultatului. In plus contine metode care adauga cate un listener fiecarui buton.

Clasele si interfata prezentate la numerele 1.-12. constituie Modelul problemei, clasa View are rolul de a desena fereastra aplicatiei (imaginea acesteia), iar clasa Controller are rolul de a face legatura dintre Model si View. Astfel, aplicatia prezinta o implementare de tip MVC, implementare ce protejeaza informatiile cu privire la modul de implementare al operatiilor.

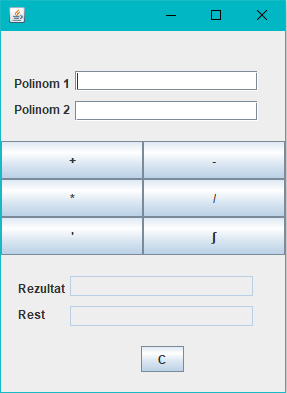
Clasele si interfata prezentate la numerele 1.-13. se regasesc in pachetul ro.tuc.tp.assig\_1, iar clasele prezentate la numerele 14.-17. fac parte din pachetul gui. Diagrama UML este :







Interfata utilizator este urmatoarea :



**4. IMPLEMENTARE**

Metodele implementate in fiecare clasa sunt :

1. Monom

* String toString( ) – returneaza monomul sub forma unui String de forma : coeficient + x^ + grad.

1. Polinom

* String toString( ) – returneaza lista de monoame care defineste polinomul sub forma unui String pentru a se afisa in interfata grafica.
* float getCoefOfDegree(int d) – returneaza coeficientul monomului care are gradul egal cu d.
* void sortPolinom( ) – metoda sorteaza monoamele unui polinom in ordine descrescatoare, prin intermediul unui comparator.
* boolean isZero( ) – metoda returneaza TRUE in cazul in care este este apelata de un polinom care are toti coeficientii egali cu 0.

1. ComparatorMonom

* int compare(Monom o1, Monom o2) – metoda stabileste modul in care se compara monoamele unui polinom pentru a se face sortarea acestuia. Metoda este declarata in interfata Comparator<Monom> si este implementata de aceasta clasa deoarece ComparatorMonom implementeaza interfata Comparator<Monom>.

1. AddOperation

* Result compute(Polinom p1, Polinom p2) – metoda declarata in interfata IOperation, interfata ce este implementata de aceasta clasa. In cadrul clasei AddOperation, metoda realizeaza adunarea polinoamelor p1 si p2 : p1+p2.

1. SubOperation

* Result compute(Polinom p1, Polinom p2) – metoda declarata in interfata IOperation, interfata ce este implementata de aceasta clasa. In cadrul clasei SubOperation, metoda realizeaza scaderea polinoamelor p1 si p2 : p1-p2.

1. MulOperation

* Result compute(Polinom p1, Polinom p2) – metoda declarata in interfata IOperation, interfata ce este implementata de aceasta clasa. In cadrul clasei MulOperation, metoda realizeaza inmultirea polinoamelor p1 si p2 : p1\*p2.

1. DivOperation

* Result compute(Polinom p1, Polinom p2) – metoda declarata in interfata IOperation, interfata ce este implementata de aceasta clasa. In cadrul clasei DivOperation, metoda realizeaza impartirea polinoamelor p1 si p2 : p1/p2. Rezultatul acestei operatii este de fapt de tipul ResultDiv, o subclasa a clasei Result, deoarece metoda trebuie sa returneze atat catul impartirii celor 2 polinoame, cat si restul.

1. DerivOperation

* Result compute(Polinom p1, Polinom p2) – metoda declarata in interfata IOperation, interfata ce este implementata de aceasta clasa. In cadrul clasei DerivOperation, metoda realizeaza derivarea polinomului p1. Al doilea parametru este in plus, fiind ignorat in corpul metodei, apare doar pentru a pastra semnatura impusa de interfata IOperation.

1. IntOperation

* Result compute(Polinom p1, Polinom p2) – metoda declarata in interfata IOperation, interfata ce este implementata de aceasta clasa. In cadrul clasei IntOperation, metoda realizeaza integrarea polinomului p1. Al doilea parametru este in plus, fiind ignorat in corpul metodei, apare doar pentru a pastra semnatura impusa de interfata IOperation.

1. ActionListenerBtn

* void actionPerformed(ActionEvent e) – metoda declarata in interfata ActionListener, interfata ce este implementata de aceasta clasa. In cadrul acestei metode sunt preluate datele de intrare si, in functie de butonul care s-a apasat (eveniment ce a apelat aceasta metoda), se executa operatia ceruta, iar rezultatul este afisat in GUI.
* IOperation getInstance(String s) – returneaza un obiect al unei clase ce implementeaza interfata IOperation, obiect ce desemneaza operatia ceruta, corespunzatoare butonului apasat de utilizator.
* void readPolinom( ) throws Exceptie – metoda citeste campul “Polinom 1”, iar in cazul in care s-a apasat un buton inainte de a se introduce un polinom in acel camp, va arunca o exceptie ce va fi tratata de metoda actionPerformed(ActionEvent e).

1. ClearListener

* void actionPerformed(ActionEvent e) – metoda declarata in interfata ActionListener, interfata ce este implementata de aceasta clasa. Aceasta metoda este apelata in momentul in care utilizatorul apasa butonul “C” si are rolul de a sterge continutul campurilor “Rezultat” si “Rest”.

1. View

* void addAddListener(ActionListener al), void addSubListener(ActionListener al),

void addMulListener(ActionListener al), void addDivListener(ActionListener al), void addDerivListener(ActionListener al), void addIntListener(ActionListener al), void addClearListener(ActionListener al) – aceste metode au rolul de a adauga cate un listener fiecarui buton.

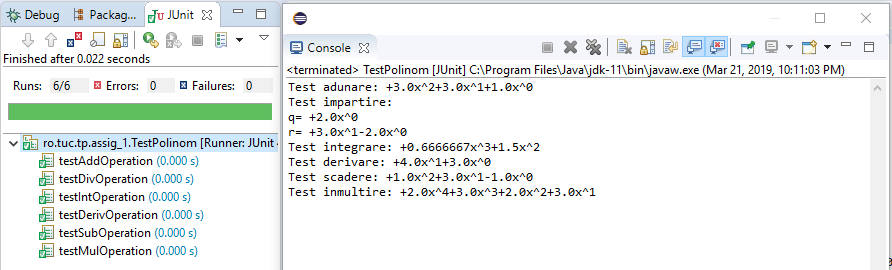
**5. REZULTATE**

Realizarea testelor s-a efectuat prin unealta de testare unitara JUnit, in cadrul clasei numite TestPolinom. Aceasta clasa are un constructor care are rolul de a initializa polinoamele si rezultatele inainte de inceperea fiecarui test. S-au implementat metodele de test : void testAddOperation( ), testSubOperation( ), testMulOperation( ), testDivOperation( ), testDerivOperation( ), testIntOperation( ), care utilizeaza metoda assertEquals(String s1, String s2) pentru a verifica daca testul a fost trecut sau nu.

Polinoamele utilizate pentru teste au fost :

P1 = 2x^2 + 3x^1

P2 = 1x^2 + 1x^0



**6. CONCLUZII**

In concluzie, aceasta tema m-a ajutat sa imi imbunatatesc cunostiintele legate de conceptele programarii orientate pe obiect. Am invatat cum pot folosi un regex pentru a extrage informatiile necesare dintr-un String, am invatat sa imi organizez mai bine codul, sa gasesc o modalitate de a scrie cat mai putin cod si cat mai clar, pentru a fi mai usor de citit, inteles si dezvoltat ulterior. In plus, am invatat sa construiesc un proiect dupa modelul MVC.

Printre posibilitatile de dezvoltare ulterioara se numara : schimbarea formatului in care se introduc polinoamele, astfel incat sa se elimine necesitatea de a scrie monoame de genul “1x^1” sau “3x^0”, in cazul in care monoamele au gradul 0 sau 1; adaugarea unor noi operatii precum calculul unui polinom intr-un punct sau aflarea radacinilor unui polinom; modificarea interfetei grafice pentru imbunatatirea aspectului.

**7. BIBLIOGRAFIE**

* <http://www.referatele.com/referate/matematica/online5/Impartirea-polinoamelor--Exemple--Impartirea-cu-rest-referatele-com.php>
* <http://www.referatele.com/referate/matematica/online1/Adunarea-si-inmultirea-polinoamelor--Proprietati-referatele-com.php>
* <https://stackoverflow.com/questions/36490757/regex-for-polynomial-expression>
* Materialele de curs si laborator de OOP