Calculator de polinoame

Duma Bianca

CTI-RO, an 2, seria B

Obiectivul temei

Scopul principal al acestei teme este implementarea unei aplicații de calcul matematic, mai exact a unui calculator de polinoame ce conține o interfață grafică prin care utilizatorul poate să introducă polinoame, să selecteze operația dorită și să vizualizeze rezultatul.

Obiective secundare

1. Analiza problemei și cazuri de utilizare……………………………………………………….3
2. Design și proiectare……………………………………………………………………………4
3. Implementarea aplicației……………………………………………………………………....7
4. Testare………………………………………………………………………………………....10
5. Concluzii......................................................................................................................................10
6. Bibliografie..................................................................................................................................11

1.ANALIZA PROBLEMEI ȘI CAZURI DE UTILIZARE

1.1Cerinte

Calculatorul de polinoame are o funcționalitate bine definită pusă în corelație cu o interfață cu utilizatorul .

a)Cerințe funcționale

În primul rând acesta oferă posibilitatea de a introduce două polinoame, de alege o operație (adunare, scădere, împărțire, înmulțire, derivare, integrare). După selectarea de către utilizator a operației dorite, aplicația poate efectua această operație, cu amendamentul că pentru derivare și integrare se va lua în considerare doar primul polinom pentru realizarea calculului. În final, rezultatul va fi afișat în forma sa corectă.

b)Cerințe non-funcționale

Interfața cu utilizatorul este alcătuită dintr-o singură fereastră și presupune caracter intuitiv și lizibilitate. Există în acest sens butoane predefinite pentru fiecare operație în parte și zone speciale de text pentru introducerea polinoamele și vizualizarea rezultatelor.

1.2 Use Case

Ca pași de urmat pentru obținerea rezultatului, operațiile de adunare, scădere, înmulțire și împărțire funcționează în mod identic, de aceea se va exemplifica doar adunarea, restul operațiilor urmând aceeași pași, singura diferență fiind bineînțeles butonul selectat.

Use Case: Adunare

Actor principal: Utilizatorul

Scenariu de utilizare:

1.Utilizatorul introduce de la tastatura doua polinoame formatate corect.

2.Utilizatorul selectează butonul specific pentru adunare.

3.Utilizatorul selectează butonul equals(“=”).

4.Se efectuează operația și se afișează rezultatul in textfield-ul cu eticheta “result”.

Secvența de pași alternativă:

-In cazul în care unul dintre polinoame sau ambele polinoame nu sunt introduse în forma lor corectă(de exemplu, apare o alta variabilă înafară de “x” sau nu se respectă formatul definit prin existența unui coeficient și a unei puteri semnalate prin caracterul “^”), pe ecran va apărea un mesaj de eroare, utilizatorul fiind rugat să se întoarcă la pasul 1 pentru a introduce o forma adecvată.

-In cazul in care utilizatorul a introdus polinoamele, însă nu a selectat operata dorita, un nou mesaj de eroare va apărea, utilizatorul fiind obligat sa selecteze unul dintre butoanele corespunzătoare operaților, deci are loc o reîntoarcere la pasul al doilea.

Cazurile enumerate mai sus urmăresc o cât mai corectă funcționare a aplicației, evitând-se astfel încă de la început eventualele erori prin semnalarea acestora.

2.DESIGN ȘI PROIECTARE

2.1Divizarea sistemului in pachete

Ca idee generală este importantă o structurare cât mai relevantă a proiectului în sine pentru o mai bună gestiune a acestuia. Această structurare impune separarea folosind pachete ca o bună practica în mediul programării orientate pe obiecte.

Așadar, un prim pachet este ”ViewPackage” ce conține clasa prin care se implementează tot ce tine de aspectul exterior al aplicației, adică partea de GUI. In pachetul “DataModels” sunt adăugate clasele ce se leagă direct de modelul datelor ales pentru dezvoltarea proiectului. La nivelul acestui pachet se vor afla astfel clasele Polynomial ,Monomial(polinomul fiind de fapt definit printr-o simplă listă de obiecte de tip monom) și PowerComparator(ce va avea rol in ordonarea descrescătoare a termenilor polinomului, aspect necesar pentru operația de împărțire) PowerComparatorDouble(corespunzător pentru monoamele cu coeficienți reali), MonomialDouble (necesar pentru operația de integrare și împărțire) și Model. Pachetul “ControllerPackage” conține clasa Controller alături de clasele interne(nested) din această clasă:AddListener, MulListener, DivListener,derListener, IntListener, ClearListener, EqualsListener,Sublistener(și tot în această clasă am adaugat și clasa Main, cea care va fi rulată). Mai există un pachet „Test” cu o clasă „ModelTest” care reprezintă unitatea de testare.

2.2Clase și arhitectura MVC

După încheierea analizei cerințelor proiectului și luarea în considerare a paradigmei programării orientate pe obiecte se ajunge la separarea codului în clase, fiecare având nume sugestive.

Pentru atingerea obiectivului proiectului se folosește arhitectura MVC ca o buna practica prin care se disting anumite avantaje.

La nivelul clasei Model sunt încapsulate datele specifice și este conturată logica aplicației. Această clasă este compusă doar din metode publice. Metodele add, substract, divide, multiply primesc ca parametrii doua obiecte de tip Polynomial, în timp ce metodele derive și integrate au cate un singur polinom ca parametru.

Clasa View întreține partea de interfața cu utilizator și este independenta de model în sensul că o schimbare la nivelul vederii nu va afecta în niciun sens modelul.

Controller este clasa prin care are loc o translatare a interacțiunilor utilizatorului cu View-ul in acțiuni pe care le va executa modelul(în cazul nostru este vorba despre click-uri de buton). Aceasta clasă interacționează atât cu modelul cât și cu vederea și dictează ce se întâmplă la click-ul fiecărui buton.

În clasa Main are loc o instantiere pentru fiecare din clasele Model, View și Controller și se permite vizibilitatea ferestrei ce cuprinde aplicația.

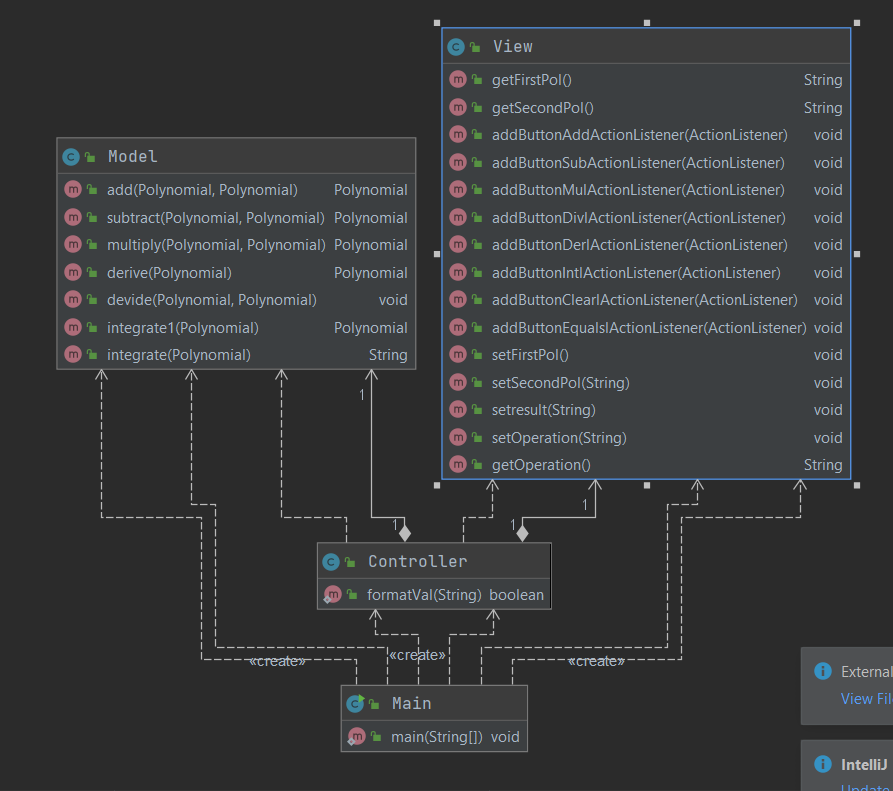
Clasa Monomial(și MonomialDouble) definește obiectul de tip termen în cadrul polinomului și operațiile care se pot implementa pe acesta, în timp ce Clasa Polynomial conturează obiectele de tip polinom. În acest sens se folosesc ca structura de date , ArrayList-uri, acestea fiind o varianta optimă pentru cerințele proiectului deoarece nu suntem nevoiți să stabilim o dimensiune fixă pentru setul de obiecte de tip monom din cadrul polinomului. De asemenea, listele de acest tip oferă metode ce facilitează anumite operații precum adăugarea și ștergerea.

Singurele interfețe folosite sunt Comparator pentru obiecte de tip Monomial(și MonomialDouble) din cadrul clasei PowerComparator(și PowerComparatorDouble) și ActionListener pentru butoanele din interfață. Prima are rol pentru a putea aplica metoda Collections.sort() care execută în spate un algoritm de sortare în concordanță cu comparatorul transmis ca al doilea parametru după lista asupra căreia se dorește a se face ordonarea. Cea de a doua este folosită în clasa Controller pentru implementarea metodei actionPerformed prin care se indica acțiunile rezultate în urma unui click pe buton.

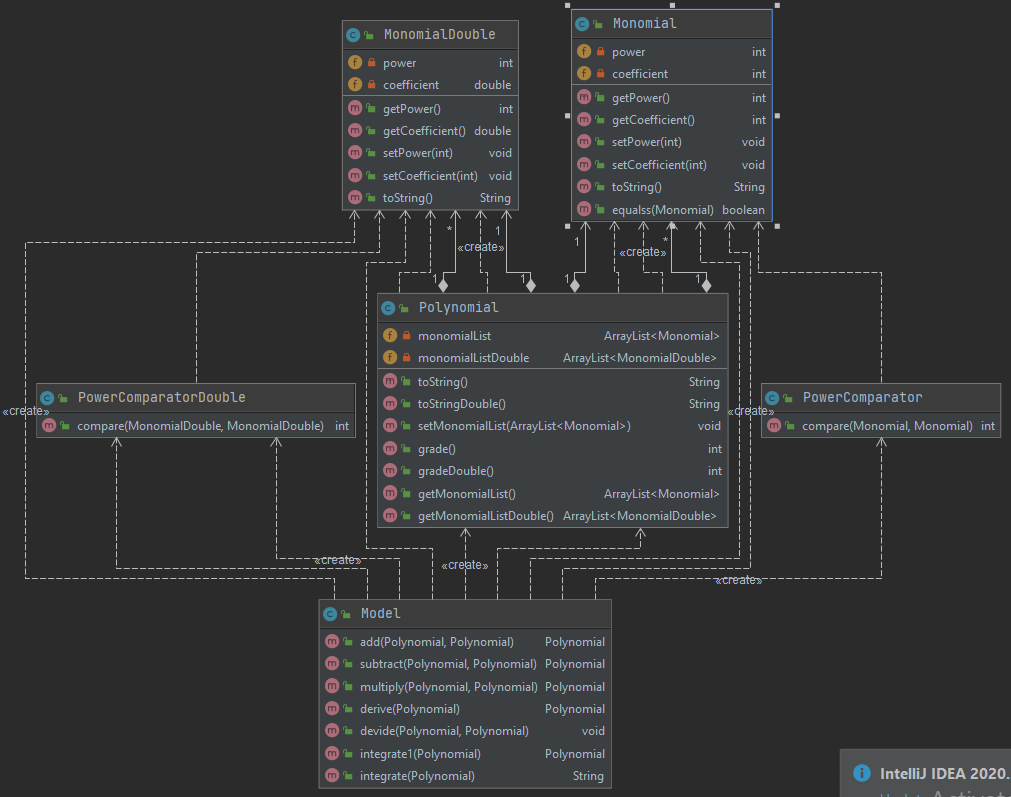
2.3 Diagrame UML

Diagramele UML sunt importante pentru reprezentarea relațiilor existente între clase(asociere, moștenire, agregare,etc). Totodată ele pun în lumină și o imagine de ansamblu a proiectarii proiectului din punct de vedere structural.

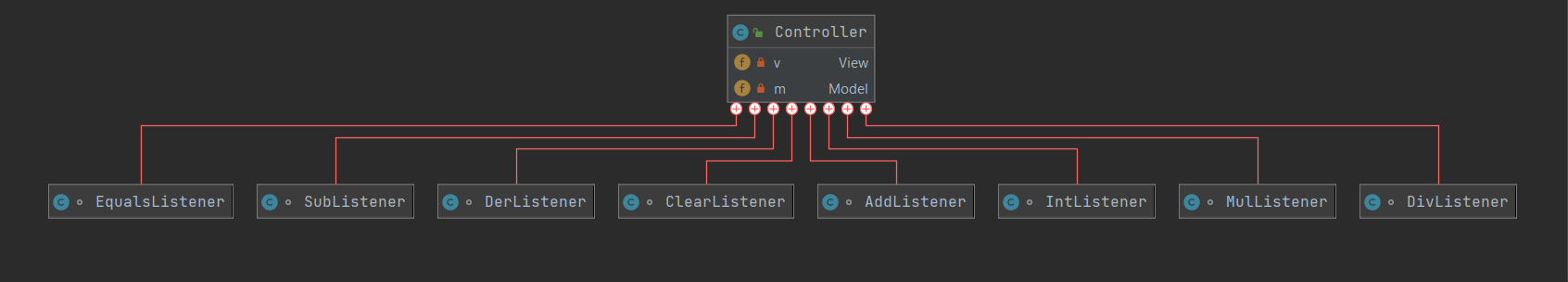
Prima diagramă ilustrează ideea arhitecturii MVC. Există o relație de compoziție(relație de tip „parte a”) între Controller si View și de asemenea între Controller și Model(cea marcată cu linie și romb), dar și relații de dependență(de tip”folosește”) între acestea. Între Model si Controller nu există nicio relație deoarece acestea sunt independente, conform principiilor de bază ale arhitecturii MVC. Am adăugat tot în acest pachet și clasa Main pentru a arăta ca aceasta se folosește de câte o instanță din fiecare dintre cele 3 clase(Model, View, Controller).



Pentru pachetul DataModels, se poate observa relația de compozițtie Intre Polynomial si cele doua clase Monomial și MonomialDouble(deoarece fiecare monom este parte a unui polinom, iar o instanță a clasei Monomial nu are sens singură). Clasa Model folosește instanțe ale clasei Polynomial, PowerComparator și PowerComparatorDouble șiMonomial și MonomialDouble.



La nivelul clasei Controller există 7 clase imbricate care implementează interfața ActionListener.



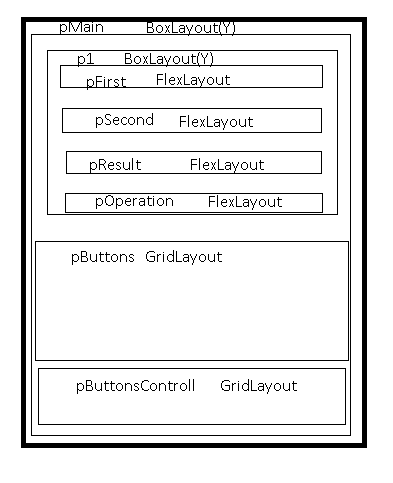
3.IMPLEMENTAREA APLICAȚIEI

Revenind la clasa Monomial, este esențial să amintim ca aceasta se definește prin existența a doua câmpuri private(idea încapsulării): power și coeficient. Ambele atribute au tipul int, iar constructorul are ca parametrii aceste valori. Potrivit paradigmei programării obiectuale, am creat și metode publice de tip set și get pentru aceste câmpuri precum și suprascrierea metodei toString pentru o afișare cât mai corecta a monomului și implicit a polinomului(pentru că acesta va apela această metoda în propria metoda toString). Am luat în considerare și cazuri speciale precum cel in care coeficientul este 0 sau când puterea este 1. Clasa MonomialDouble este identica, însa aici coeficienții sunt de tip real.

Clasa Polynomial are două câmpuri private și anume MonomialList și MonomialListDouble care sunt de tip ArrayList. Metoda constructului este mai complexă decât la clasa Monomial. Parametrul este de tip String deoarece la nivelul vederii, prin textfield se transmite un astfel de tip de date. Mai departe, am folosit aspecte precum regex, pattern matching și substring-uri pentru extragerea parților componente pentru formarea monoamelor pe care le- am adăugat apoi în lista de monoame a polinoamelor, la final rezultând lista întreaga de termini prin care se definește practic obiectul de tip polinom. Metodele acestei clase sunt de asemenea publice, fiind vorba de suprascrierea metodei toString care va returna un șir de caractere și o metoda suplimentara care returnează ca număr întreg ordinul polinomului pentru care am folosit un algoritm de aflare a maximului în cadrul listei de monoame.

După cum am amintit și mai sus, clasa Model implementează metodele publice de adunare, scădere, înmulțire, împărțire, derivare și integrare. Respectând convențiile matematice, am dezvoltat codul pentru fiecare dintre acestea, folosindu-mă de traversarea listelor cu for each și de metode precum add și remove. De asemenea pentru o mai buna lizibilitate am aplicat și o sortare a monoamelor din polinom pentru a obține o listă descrescătoare din punct de vedere al puterii acestora. Pentru adunare, se parcurg polinoamele, iar atunci când se găsesc doua monoame care au aceeași putere, coeficientul primului se setează ca fiind suma dintre propriul coeficient și coeficientul monomului din al doilea polinom(rezultatul se salvează în polinomul de ordin mai mare). Pentru a evita o repetare în scriere adăugam monomul din al doilea polinom într-o listă pe care o vom prelucra la final(în cazul în care suntem pe ramura if a buclei interioare, adică dacă am găsit monoame de aceeași putere) .La ieșirea din bucle, ce mai rămâne de făcut este să adunăm monoamele care au rămas în cel de-al doilea polinom, adică cele care nu și-au găsit corespondent ca putere .Pentru ajungerea la un rezultat corect, mai întâi scoatem din lista de monoame a polinomului al doilea toate monoamele care “și-au găsit corespondent”, iar apoi adunăm această listă la rezultat. În cazul scăderii, codul este absolut similar, cu amendamentul că schimbăm semnul pentru monoamele din polinomul al doilea. Pentru înmulțire, procesul se rezumă la înmulțirea fiecărui monom din primul polinom cu fiecare monom din al doilea polinom. Se setează astfel coeficientul rezultat ca fiind produsul coeficienților, iar puterea rezultata ca fiind suma puterilor. În a treia bucla se verifica dacă există puteri asemenea pentru a se face adunarea. Se ia în considerarea și ștergerea termenilor din rezultat pentru evitarea rezultatelor de genul 2x^3+4x^3, iar monomul final se adaugă la rezultat. Derivarea și integrarea presupun procesarea pe rând a fiecărui monom în parte și adăugarea lor la rezultatul final. La integrare trebuie luat in calcul posibilitatea scrierii coeficientului de tip realși adăugarea constantei “c” la finalul rezultatului. Pentru operația de împărțire este necesara returnarea rezultativului alături de rest, ambele fiind polinoame. La început are loc sortarea după puteri în ordine descrescătoare, iar dacă împărțitorul este diferit de 0 atunci se poate executa împărțirea(se împarte cel cu gradul mai mare la cel cu gradul mai mic).Primul monom al deîmpărțitului se împarte la primul monom al împărțitorului și astfel se obține primul termen al câtului. Câtul se înmulțește cu deîmpărțitul și se scade din împărțitor, obținând-se un rest, iar procesul se repetă atât timp cat restul este diferit de 0 și gradul restului este mai mare decât gradul împărțitorului.

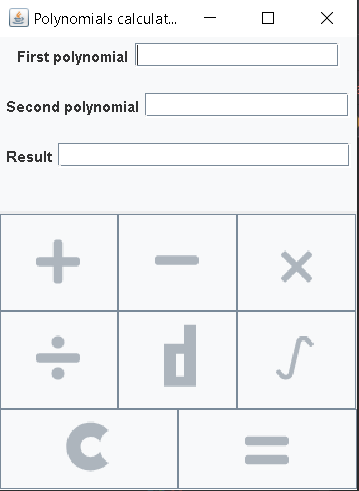
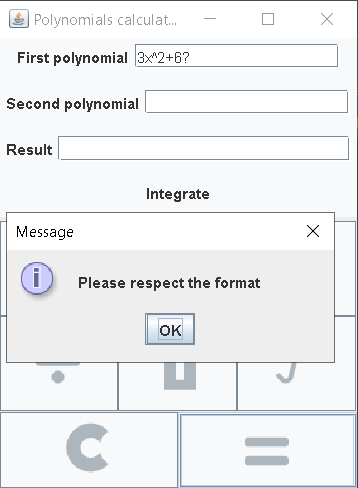
Clasa View conține o multitudine de câmpuri care reprezintă elementele ce formează interfața. Este vorba de mai multe câmpuri private pentru butoane, JButtons,field-uri pentru text(JTextField), etichetele JLabel și de asemenea am folosit și image icons pentru o afișare mai aspectuoasa. In constructor, am setat dimensiuni, am utilizat mai multe panouri JPanel , iar ca aranjare am utilizat atât BoxLayout și FlexLayout(acesta este implicit) cât și GridLayout pentru butoane. Exista numeroase metode publice și anume: de tip get și de tip set pentru textfield-uri și etichete, dar și adăugarea de ascultători pentru fiecare buton in parte.



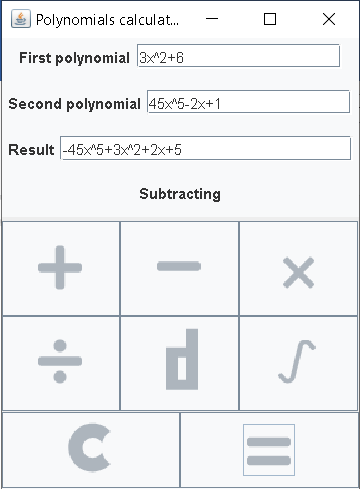
După cum evidențiază și numele, controlul aplicației se face mai ales prin clasa Controller care are ca atribute private un model și o vedere . Constructorul clasei are doi parametrii de acest tip, iar pentru fiecare buton există o clasă care implementează interfața ActionListener a cărei metoda obligatorie este actionPerformed().Ceea ce se face de fapt este setarea etichetei cu numele operației alese. La apăsarea butoanelor specifice operaților, eticheta Jlabel din partea de sus a butoanelor se va actualiza cu numele operației alese. Butonul clear(C) are rolul de a elibera toate cele 3 textfield-uri, având evident rol de reset. Cel mai important buton este însă cel pentru afișarea rezultatul(“=”) prin care se comanda de fapt înainte efectuarea operației dorite. Astfel, un click pe acest buton va determina mai departe următoarele: verificarea unei formatări corecte a polinoamelor și a selecției în prealabil a operației. În cazul in care aceste doua condiții sunt îndeplinite, se poate efectua operația(prin apelul unei metode din Model) aleasă asupra polinoamelor și se afișează rezultatul, apelând o metoda de setare a textfield-ului pentru rezultat din View.

PowerComparator (respectiv PowerComparatorDouble)este clasa care implementează interfața Comparator<Monomial>(respectiv Comparator<MonomialDouble>) și are o singură metodă, compare cu un argument care returnează un întreg în funcție de condițiile specificate.

Imagini cu interfața:

1. Interfața la pornirea aplicației 2.Interfața în caz de introducere a unui polinom nevalid



3.Interfața afișând un rezultat corect

4.TESTARE

Testarea este unul dintre cei mai importanți pași în dezvoltarea software, fiind un proces decisiv și absolut necesar pentru orice program. Astfel se pot detecta eventualele erori sau nereușita îndeplinirii uneia sau mai multor cerințe inițiale ori coruperea acestora. Exista mai multe moduri și tehnologii de testare, dar cea aleasă pentru acest proiect este JUnit.

Ca idee, testarea unitara se referă la posibilitatea testării individuale a unor entități(unități) separate dintr-un program. Aceste unități pot sa fie spre exemplu, metodele.

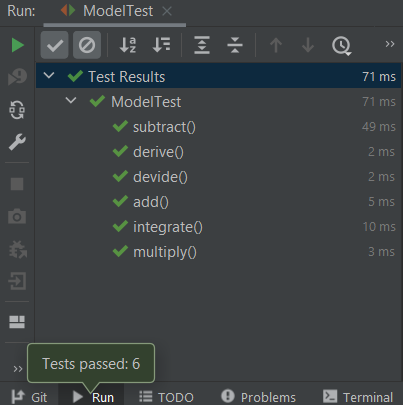
4.1Rezultate și JUnit

Pentru testarea acestui proiect, s-a creat mai întâi un pachet separat intitulat “Test”. Clasa din interiorul acestui pachet este ModelTest, unde am importat pachetul org.junit.jupiter.Assertions.\*. Ca atribute private, există două polinoame(p1 și p2), un String intitulat operation prin care se va puncta care dintre operații este testată și o instanță de Model pentru a putea apela funcțiile care execută calculele matematice.

Există două metode importante și anume setUp care se execută înaintea fiecărui test și tearDown care se execută după fiecare test. La nivelul primei funcții sunt mereu inițializate atributele clasei și se afișează un mesaj corespunzător, iar la nivelul metodei tearDown de asemenea se afișează un mesaj prin care se face înțeles faptul că testul este corect.

Pentru fiecare operație în parte există o metodă precedata de adnotarea @org.junit.jupiter.api.Test în care se face testul pentru diferite polinoame. Rezultatul este comparat de fiecare dată cu rezultatul așteptat utilizând assertTrue.

Astfel, spre exemplu adunând polinomul 2x^3+5x^4+2x cu polinomul 5x^4+2x se va obține într-adevăr 10x^4+2x^3+4x.



5.CONCLUZII

Acest proiect realizează o aplicație matematică utilă ce reunește numeroase concepte de programare orientate pe obiect pentru obținerea unei funcționalități corecte, lucru care va fi sumarizat mai departe.Utilizând arhitectura MVC ca suport fundamental, se construiește un cod unde interfața grafica este independentă de modelul matematic intern(metodele pentru operații). Cu ajutorul pachetului swing interfața grafică se prezintă ca una ușor de folosit pentru orice utilizator, prin existența panourilor cu butoane, zone pentru text, dar și posibilitatea aranjării acestora prin diferite forme de layout. Interacțiunea directă cu aplicația se face prin implementarea metodelor din clasa Controller unde esențiali sunt ascultătorii, concept de bază in aplicațiile Java, având in vedere deci și interfața ActionListener. Un alt aspect este utilizarea expresiilor regulate alături de “pattern matching” pentru formatarea corecta a polinoamelor. Un rol important a fost ocupat și de existenta unei clase de tip Comparator care implementează interfața Comparable pentru a putea folosi metoda sort a clasei Collections. O alta resursă de menționat în cadrul procesului de realizare a acestui proiect este componenta JUnit pentru partea de testare.

La toate acestea se adaugă structurarea claselor și a pachetelor și respectarea conceptelor și convențiilor Java cunoscute.

6.Bibliografie

1. <http://users.utcluj.ro/~igiosan/Resources/POO/Lab/12-Testarea_Unitara.pdf>
2. <http://tutorials.jenkov.com/java-regex/matcher.html>
3. <http://users.utcluj.ro/~igiosan/Resources/POO/Curs/POO06.pdf>