Documentație de temă

Calculator Polinomial

Proiect realizat de: Anton Ioana-Carina

Grupa 30225

Cuprins

[1. Obiectivul temei 4](#_Toc67442690)

[Obiectivul principal 4](#_Toc67442691)

[Obiective secundare 4](#_Toc67442692)

[2. Analiza problemei 4](#_Toc67442693)

[Diagramele use-case 4](#_Toc67442694)

[Cerinte functionale 7](#_Toc67442695)

[Cerinte non-functionale 7](#_Toc67442696)

[3. Decizii de proiectare 7](#_Toc67442697)

[Arhitectura MVC 7](#_Toc67442698)

[Structuri de date 7](#_Toc67442699)

[Cadrul de lucru Java Collections 7](#_Toc67442700)

[Interfata Collection 8](#_Toc67442701)

[Clasa ArrayList 8](#_Toc67442702)

[Diagrama UML 8](#_Toc67442703)

[Packages 8](#_Toc67442704)

[Pachetul Model 8](#_Toc67442705)

[Pachetul View 9](#_Toc67442706)

[Pachetul Controller 9](#_Toc67442707)

[4. Implementare 9](#_Toc67442708)

[Controller 9](#_Toc67442709)

[Variabile instanta 9](#_Toc67442710)

[Metode si subclase 9](#_Toc67442711)

[ComparatorDegree 9](#_Toc67442712)

[Metoda compare 9](#_Toc67442713)

[Model 9](#_Toc67442714)

[Variabile instanta 9](#_Toc67442715)

[Metode 9](#_Toc67442716)

[Monomial 10](#_Toc67442717)

[Variabile instanta 10](#_Toc67442718)

[Metode 10](#_Toc67442719)

[Polynomial 10](#_Toc67442720)

[Variabile instanta 10](#_Toc67442721)

[Metode 10](#_Toc67442722)

[View 10](#_Toc67442723)

[Variabile instanta 10](#_Toc67442724)

[Metode 11](#_Toc67442725)

[5. Rezultate 11](#_Toc67442726)

[Adunare 11](#_Toc67442727)

[Scadere 12](#_Toc67442728)

[Inmultire 13](#_Toc67442729)

[Impartire 14](#_Toc67442730)

[Derivare 15](#_Toc67442731)

[Integrare 15](#_Toc67442732)

[Rezultate testare 16](#_Toc67442733)

[6. Concluzii 16](#_Toc67442734)

[Dezvoltari ulterioare 16](#_Toc67442735)

[Ce am invatat 16](#_Toc67442736)

[7. Bibliografie 16](#_Toc67442737)

# Obiectivul temei

## Obiectivul principal

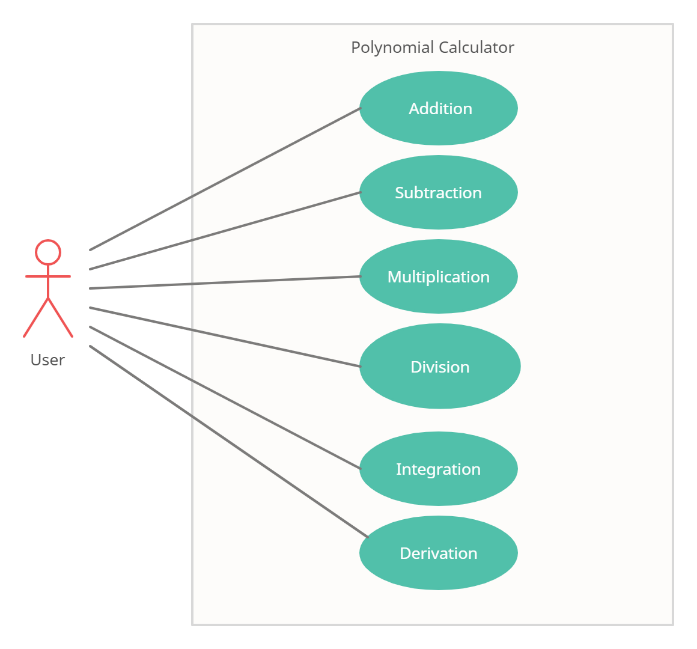
Obiectivul temei este de a proiecta si implementa un calculator de polinoame cu o interfata grafica dedicata prin care utilizatorul poate sa introduca doua polinoame, sa selecteze operatia dorita ( adunare, scadere, inmultire, impartire, derivare si integrare) si sa vizualizeze rezultatul.

## Obiective secundare

|  |  |  |
| --- | --- | --- |
| **Analiza problemei si identificarea cerintelor** | Se stabileste modul de functionare al aplicatiei, se construiesc diagramele use-case si se stabilesc cerintele functionale, respectiv nefunctionale ale problemei | Se detalieaza in capitolul 2 |
| **Proiectarea calculatorului de polinoame** | Se stabileste structura modelului dupa care se va construi programul, se stabilesc pachetele, clasele, relatiile dintre acestea (Diagrama UML), etc. | Se detaliaza in capitolul 3 |
| **Implementarea calculatorului de polinoame** | Se transcriu in cod efectiv diagramele de la etapa de proiectare | Se detaliaza in capitolul 4 |
| **Testarea calculatorului de polinoame** | Se utilizeaza un framework de testare pentru a verifica functionalitatea aplicatiei si cazurile acoperita de acestea | Se detaliaza in capitolul 5 |

# Analiza problemei

## Diagramele use-case



**Use Case**: Adunare

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1.Utilizatorul introduce doua polinoame in interfata grafica

2.Utilizatorul selecteaza operatia de adunare

3.Utilizatorul apasa butonul de confirm

4.Calculatorul polinomial face adunarea celor doua polinoame si afiseaza rezultatul

**Alternative Sequence**: Incorrect polynomials

- Utilizatorul introduce polinoamele in mod gresit

- Scenariul se intoarce la pasul 1

**Use Case**: Scadere

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1.Utilizatorul introduce doua polinoame in interfata grafica

2.Utilizatorul selecteaza operatia de scadere

3.Utilizatorul apasa butonul de confirm

4.Calculatorul polinomial face scaderea celor doua polinoame si afiseaza rezultatul

**Alternative Sequence**: Incorrect polynomials

- Utilizatorul introduce polinoamele in mod gresit

- Scenariul se intoarce la pasul 1

**Use Case**: Inmultire

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1.Utilizatorul introduce doua polinoame in interfata grafica

2.Utilizatorul selecteaza operatia de inmultire

3.Utilizatorul apasa butonul de confirm

4.Calculatorul polinomial face inmultirea celor doua polinoame si afiseaza rezultatul

**Alternative Sequence**: Incorrect polynomials

- Utilizatorul introduce polinoamele in mod gresit

- Scenariul se intoarce la pasul 1

**Use Case**: Impartire

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1.Utilizatorul introduce doua polinoame in interfata grafica

2.Utilizatorul selecteaza operatia de impartire

3.Utilizatorul apasa butonul de confirm

4.Calculatorul polinomial face impartirea celor doua polinoame si afiseaza rezultatul

**Alternative Sequence**: Incorrect polynomials

- Utilizatorul introduce polinoamele in mod gresit

- Scenariul se intoarce la pasul 1

**Use Case**: Derivare

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1.Utilizatorul introduce un polinom in interfata grafica

2.Utilizatorul selecteaza operatia de derivare

3.Utilizatorul apasa butonul de confirm

4.Calculatorul polinomial deriveaza polinomul introdus si afiseaza rezultatul

**Alternative Sequence**: Incorrect polynomials

- Utilizatorul introduce polinomul in mod gresit

- Scenariul se intoarce la pasul 1

**Use Case**: Integrare

**Primary Actor**: utilizatorul

**Main Success Scenario**:

1.Utilizatorul introduce un polinom in interfata grafica

2.Utilizatorul selecteaza operatia de integrare

3.Utilizatorul apasa butonul de confirm

4.Calculatorul polinomial integreaza polinomul introdus si afiseaza rezultatul

**Alternative Sequence**: Incorrect polynomials

- Utilizatorul introduce polinomul in mod gresit

- Scenariul se intoarce la pasul 1

## Cerinte functionale

* Calculatorul polinomial ar trebui sa ii permita utilizatorului sa introduca doua polinoame.
* Calculatorul polinomial ar trebui sa ii permita utilizatorului sa isi aleaga operatia dorita.
* Calculatorul polinomial ar trebui sa adune doua polinoame.
* Calculatorul polinomial ar trebui sa scada doua polinoame.
* Calculatorul polinomial ar trebui sa inmulteasca doua polinoame.
* Calculatorul polinomial ar trebui sa imparta doua polinoame.
* Calculatorul polinomial ar trebui sa integreze un polinom.
* Calculatorul polinomial ar trebui sa deriveze un polinom.
* Calculatorul polinomial ar trebui sa anunte utilizatorul daca a introdus unul dintre cele doua polinoame nu au fost scrise corect.
* Calculatorul polinomial ar trebui sa anunte utilizatorul daca nu si-a ales operatia.

## Cerinte non-functionale

* Calculatorul polinomial ar trebui sa aiba o interfata grafica atractiva, intuitiva si usor de utilizat.

# Decizii de proiectare

## Arhitectura MVC

Ca si arhitectura principala a programului am ales arhitectura Model-View-Controller (MVC) pentru a avea o structura usor de utilizat si bine organizata.

**Modelul** incapsuleaza datele specifice aplicatiei, defineste logica si computatiile care manipuleaza si proceseaza datele respective. Modelul nu stie despre vederi si controloare. Cand se schimba, in mod tipic modelul notifica observatorii sai despre schimbare.

**Vederea** reda continutul unui model. Specifica exact cum se prezinta utilizatorului datele din model. Daca modelul se schimba, vederea trebuie sa-si actualizeze prezentarea dupa nevoi.

**Controlorul** traduce interactiunile utilizatorului cu vederea in actiuni pe care la va executa modelul.

## Structuri de date

### Cadrul de lucru Java Collections

Un cadru de lucru este o arhitectura unificata destinata reprezentarii si manipularii colectiilor. Toate cadrele de lucru din JCF (Java Collection Framework) contin:

* **Interfete** – tipuri de date abstracte care reprezinta colectiile si care permit ca respectivele colectii sa fie manipulate independent de detaliile de implementare
* **Implementari** – implementari concrete ale interfetelor colectiilor. Sunt, in esenta, structuri de date reutilizabile
* **Algoritmi** – metode care efectueaza operatii utile, cum sunt sortarea si cautarea, pe/in obiecte care implementeaza interfetele. Algoritmii sunt polimorfi: adica, aceeasi metoda se poate folosi pe implementari diferite ale interfetei colectie corespunzatoare. Algoritmii sunt functionalitate reutilizabila.

Cateva beneficii ale acestui cadru le lucru:

* **Reducerea efortului de** programare – prin furnizarea de structuri de date si algoritmi, utile in dezvoltarea aplicatiilor.
* **Cresterea vitezei si calitatii programelor** – implementarile sun tinterschimbabile, astefel ca programele pot fi adjustate prin schimbarea implementarilor colectiilor
* **Favorizarea reutilizarii software** – noile structuri de date care se conformeaza interfetelor colectie standard sunt natural reutilizabile, lucru valabil si pentru algoritmii care opereaza pe obiectele care implementeaza respectivele interfete

### Interfata Collection

Interfata Collection modeleaza un grup de obiecte numire elemente. Scopul acestei interfete este sa faciliteze folosirea colectiilor la nivel maxim de generalitate.

Aceasta interfata contine trei categorii de metode:

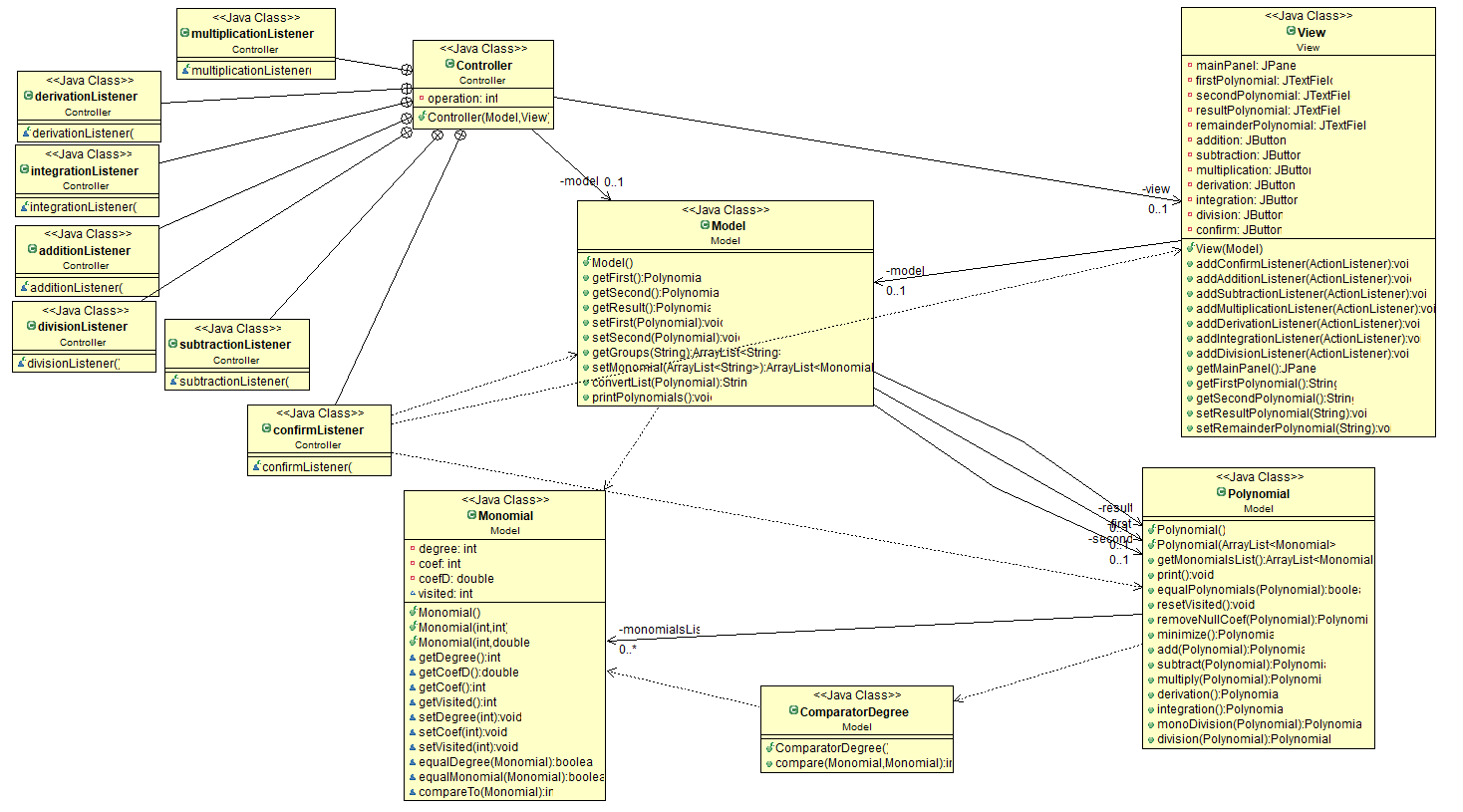
* **Operatii la nivel de element** : isEmpty(), contains(E e), add(E e), remove(E e), iterator()
* **Operatii la nivel de colectie**: size(), containsAll(Collection<E> c), addAll(Collection<E> c), removeAll(Collection<E> c), retainAll(Collection<E> c), clear()
* **Operatii de conversie in tablou unidimensional:** toArray()

### Clasa ArrayList

Principala clasa din cadrul JCF utilizata in proiect este ArrayList. Clasa ArrayList implementeaza o secventa de valori stocate intr-un tablou re-dimensionabil.

Pentru implementarea metodei sort, a fost utilizata o clasa auxiliara care implementeaza Comparator.

## Diagrama UML



## Packages

Programul a fost impartit in 3 pachete diferite, dupa modelul arhitecturii MVC.

### Pachetul Model

Pachetul corespunzator modelului contine 4 clase: Monomial, Polynomial, Model si ComparatorDegree.

**Clasa Monomial** contine structura de baza a datelor reprezentate de polinom, adica structura unui monom si operatiile facute la nivel de unitate de monom.

**Clasa Polynomial** reprezinta modelul principal de date al problemei, un polinom reprezentat ca si o insiruire de monoame. In aceasta clasa sunt incluse si operatiile specifice pe polinoame.

**Rolul clasei model** este de a oferi posibilitatea de comunicare dintre datele introduse ca si input de catre utilizator si convertirea lor in modelul de date utilizat de program, adica convertirea datelor introduse ca si text in polinoame.

**Clasa ComparatorDegree** implementeaza interfata Comparator pentru a putea realiza o sortare in cadrul monoamelor dintr-un polinom.

### Pachetul View

Acest pachet contine o singura clasa numita **View** in care a fost realizata interfata grafica (GUI) utilizand JavaSwing.

### Pachetul Controller

Pachetul Controller contine de asemenea o singura clasa denumita **Controller**. Aceasta clasa contine insa multiple subclase care faciliteaza comunicarea dintre actiunile utilizatorului si program.

# Implementare

## Controller

### Variabile instanta

* **Private Model model** = reprezinta modelul principal din arhitectura MVC.
* **Private View view** = reprezinta vederea principala din arhitectura MVC.
* **Private int operation** = o variabila simpla intreaga pentru a monitoriza operatia dorita de catre utilizator.

### Metode si subclase

* **Public Controller(Model m, View v)** = constructorul clasei, definit cu acces public pentru a putea fi accesat din afara pachetului de catre functia main; in constructor se adauga si ActionListenerii la vedere.
* **Class confirmListener** = subclasa care implementeaza interfata ActionListener; reprezinta subclasa principala in care se iau datele input din interfata iar pe baza variabilei operation se alege operatia care se executa de catre program la momentul apasarii butonului de confirm si afisarea rezultatului in interfata.
* Fiecare dintre subclasele ramase implementeaza interfata ActionListener pentru fiecare buton de operatie din interfata utilizator si schimba valoarea variabilei operation in cea specifica operatiei selectate de catre utilizator.

## ComparatorDegree

### Metoda compare

Singura metoda inclusa in aceasta clasa, clasa care implementeaza interfata Comparator<Monomial>. Aceasta metoda compara doi monomi.

## Model

### Variabile instanta

* **Private Polynomial first**
* **Private polynomial second**
* **Private polynomial result**

Cele trei variabile instanta reprezinta datele principale pe care se executa operatiile de polinoame, respectiv rezultatul lor.

### Metode

* **public ArrayList<String> getGroups(String s)** = primeste ca parametru un polinom sub forma unui sir de caractere si separa monoamele din el dupa semne si le pune intr-un ArrayList de string uri
* **public ArrayList<Monomial> setMonomial(ArrayList<String> s)** = primeste ca parametru ArrayList ul de monoame sub forma de stringuri si le transforma intr-un ArrayList de monoame, introducand la fiecare coeficientul si gradul
* **public String convertList(Polynomial p)** = primeste ca parametru polinomul rezultat din urma operatiei si il converteste inapoi intr-un polinom sub forma de string pentru a putea fi afisat in interfata utilizator

## Monomial

### Variabile instanta

* **private int degree** = gradul monomului;
* **private int coef** = coeficientrul intreg al monomului;
* **private double coefD** = coeficientul real al monomului rezultat dupa integrare;
* **int visited** = variabila utilizata la operatii precum adunarea unde nu toti termenii vor face parte din calculul rezultatului final; asa ca cu ajutorul ei ne asiguram ca vor aparea in polinomul final si termenii care nu au fost vizitati de catre operatie;

### Metode

* 3 constructori diferiti, unul fara parametrii, unui cu parametrii intregi si unu pentru polinoamelele cu coeficienti reali
* **boolean equalDegree(Monomial a)** = returneaza true daca gradele monoamelor sunt egale
* **boolean equalMonomial(Monomial a)** = returneaza true daca atat gradul cat si coeficientul monoamelor sunt egale
* **int compareTo(Monomial p)** = metoda necesara pentru metoda compare din clasa ComparatorDegree pentru a ordona monoamele dupa grad

## Polynomial

### Variabile instanta

* **private ArrayList<Monomial> monomialsList** = singura variabila instanta care reprezinta sirul de monoame din care este alcatuit polinomul

### Metode

* Doi constructori, unul fara parametrii si unui pentru initializarea ArrayList ului de monoame
* **public boolean equalPolynomials(Polynomial a)** = returneaza true daca cele doua polinoame sunt egale
* **public void resetVisited()** = reseteaza variabila visited la 0 la toate monoamele din polinom
* **public Polynomial removeNullCoef(Polynomial a**) = elimina din polinom monoamele care au coeficient 0
* **public Polynomial minimize()** = aduce polinomul la o forma finalizata, adunand monoamele care au acelasi grad
* **public Polynomial add(Polynomial a)** = efectueaza adunarea dintre doua polinoame; parcurge cu for in for cele doua polinoame si aduna termenii de acelasi grad dintre cei doi si ii adauga intr-un polinom rezultat, se adauga la final si termenii care nu au fost vizitati in bucla si apoi se returneaza polinomul final
* **public Polynomial subtract(Polynomial a)** = efectueaza operatie de scadere dintre doua polinoame; procedeul este similar cu cel de la adunare, numai ca in loc sa se adune coeficientii monoamelor de acelasi grad, se scad
* **public Polynomial multiply(Polynomial a)** = efectueaza operatia de inmultire dintre doua polinoame; se multeste fiecare monom dintr-un polinom cu fiecare monom din celalalt utilizand un for in for, gradele sunt adunate, iar coeficientii se inmultesc
* **public Polynomial derivation()** = efectueaza operatia de derivare asupra unui polinom, se parcurge fiecare monom din polinomul initial si se scade gradul monomului, iar coeficientul este inmultit cu gradul initial; monoamele din polinomul initial care au gradul 0 nu sunt adaugate in polinomul rezultat
* **public Polynomial integration()** = efectueaza operatia de integrare asupra unui polinom, se parcurge fiecare monom din polinomul initial si se creste gradul cu o unitate, iar coeficientul este impartit la gradul nou si monomul rezultat are coeficientul de tip double
* **public Polynomial[] monoDivision(Polynomial a)** = contine pasii din cadrul operatiei de impartire a polinoamelor, va fi retunat catul si restul
* **public Polynomial[] division(Polynomial a)** = metoda completa care efectueaza operatie de impartire, apeland metoda monoDivision in functie de care polinom are gradul mai mare

## View

### Variabile instanta

* Doua variabile JTextField editabile pentru introducerea polinoamelor
* Doua variabile JtextField needitabile pentru afisarea rezultatului si restului operatiilor
* Butoane pentru fiecare operatie care se poate efectua
* Un buton de confirm pentru a efectua operatia aleasa

### Metode

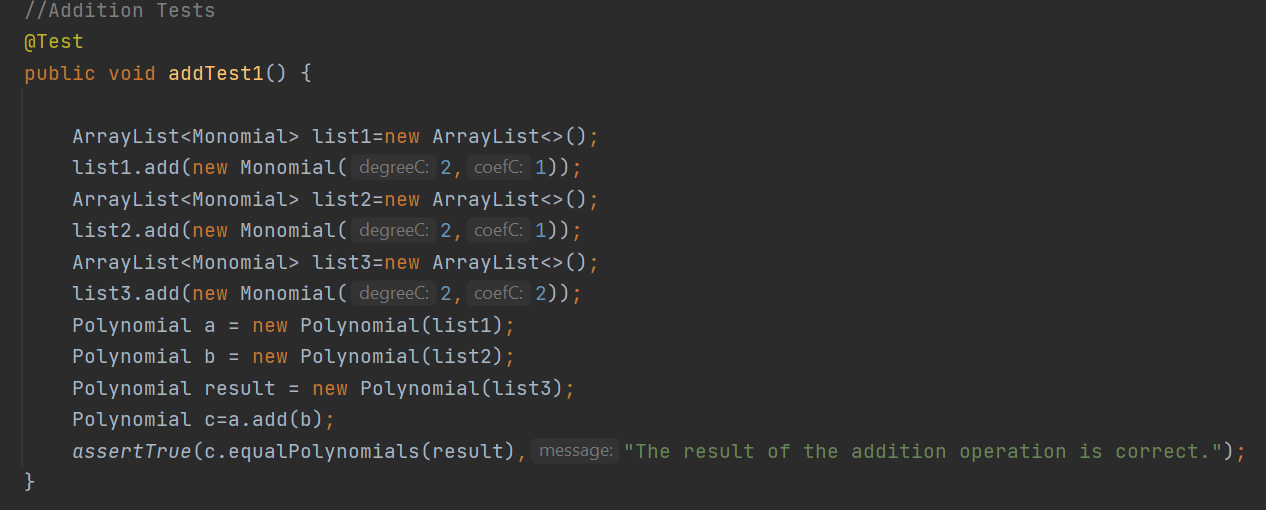
* Constructorul = primeste ca parametru modelul; se creaza interfata utilizator alegand dimensiunile potrivite pentru fiecare componenta si adaugarea lor la Jpanelul principal
* Metode pentru adaugarea de ActionListeneri la butoane
* Cate un getter pentru polinoamele introduse de catre utilizator
* Doi setteri pentru JTextField ul rezultatului, repsectiv restului

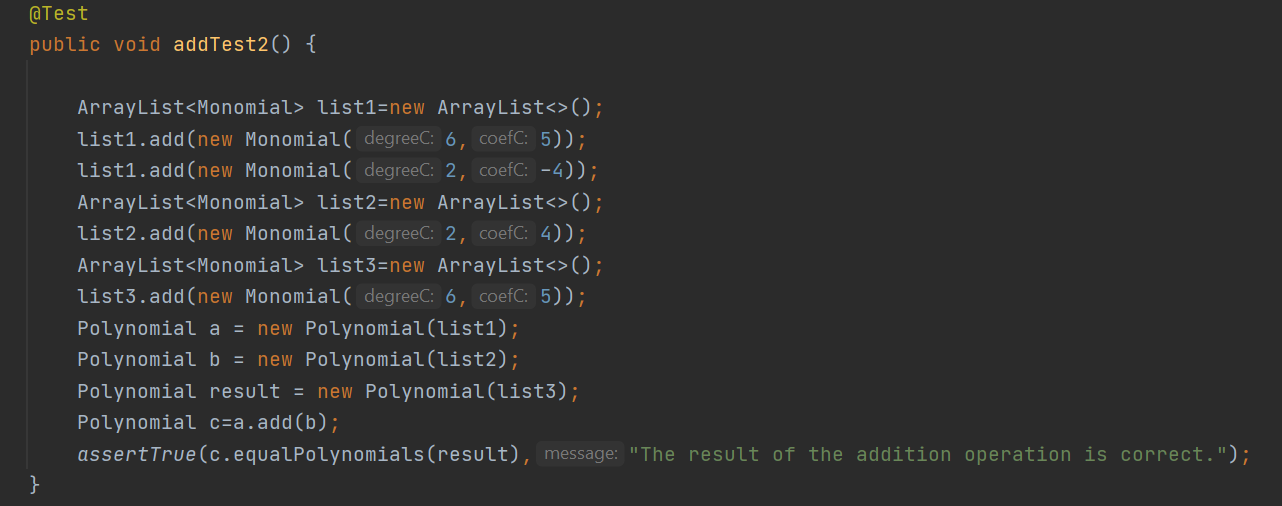
# Rezultate

## Adunare

Polinoamele pe care s-au facut testele:

* Test 1: P(x) = 1x^2 si Q(x) = 1x^2 => C(x) = 2x^2
* Test 2: P(x) = 5x^6-4x^2 si Q(x) = 4x^2 => C(x) = 5x^6
* Cazurile de testare: adunare obisnuita, adunare si eliminare termen coeficient 0

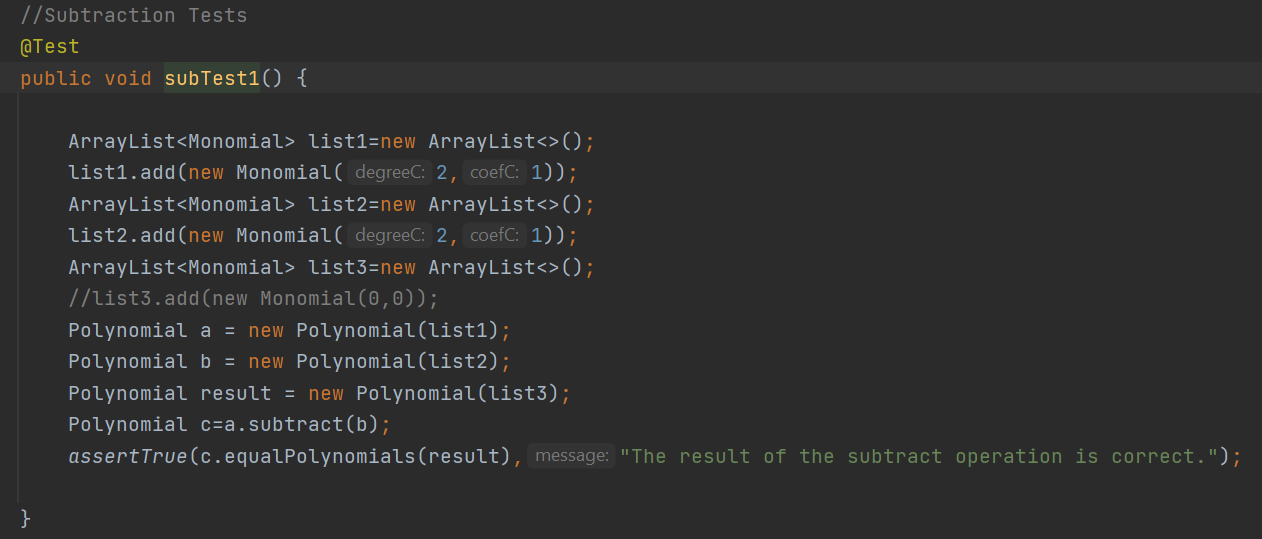


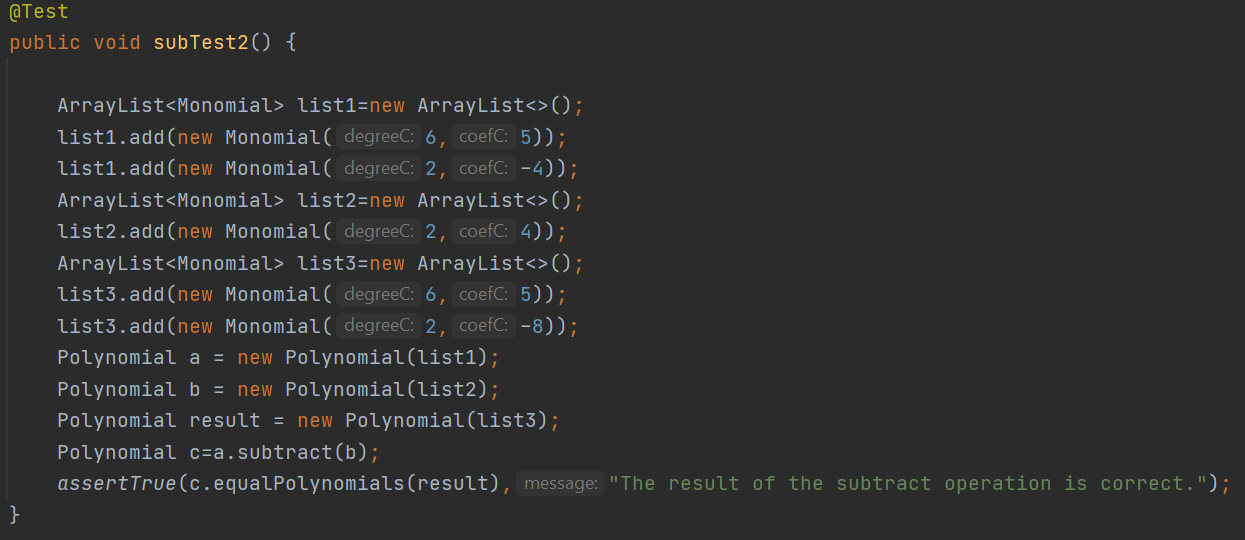


## Scadere

Polinoamele pe care s-au facut testele:

* Test 1: P(x) = 1x^2 si Q(x) = 1x^2 => C(x) = 0
* Test 2: P(x) = 5x^6-4x^2 si Q(x) = 4x^2 => C(x) = 5x^6-8x^2
* Cazurile de testare: verificarea eliminarii termenilor cu coeficient 0, scadere obisnuita

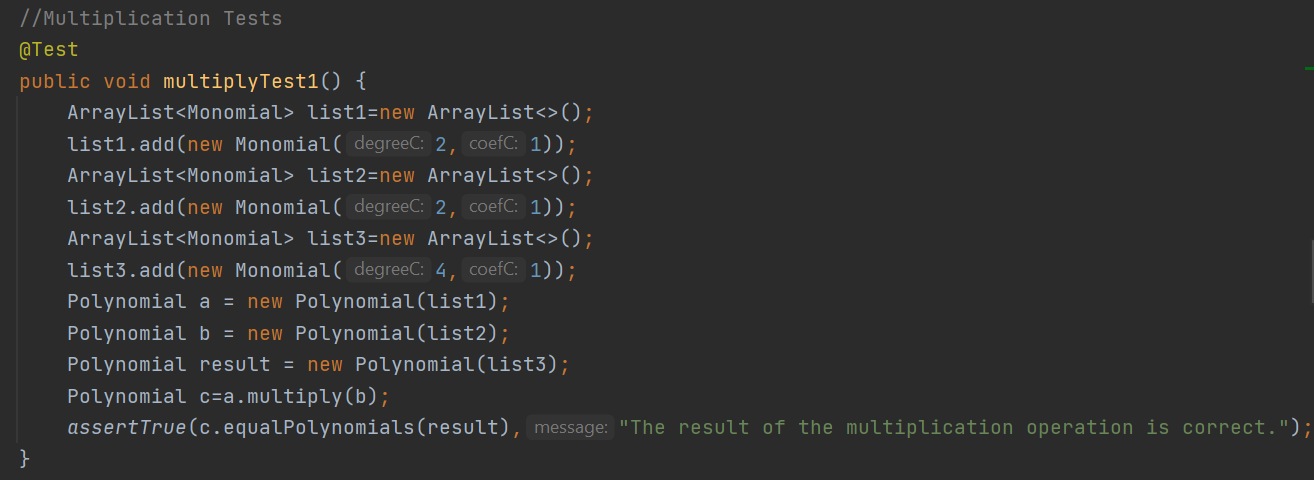


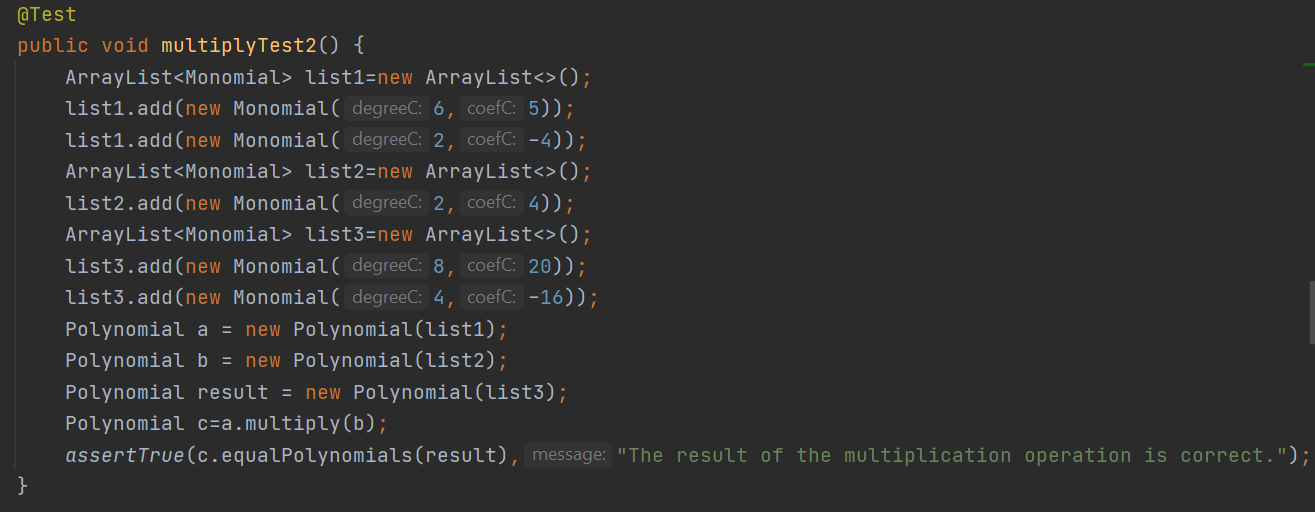


## Inmultire

Polinoamele pe care s-au facut testele:

* Test 1: P(x) = x^2 si Q(x) = x^2 => C(x) = 1x^4
* Test 2: P(x) = 5x^6-4x^2 si Q(x) = 4x^2 => C(x) = 20x^8-16x^4
* Cazurile de testare: pentru primul test se verifica si functia de minimize(), la al doilea inmultirea termen cu termen

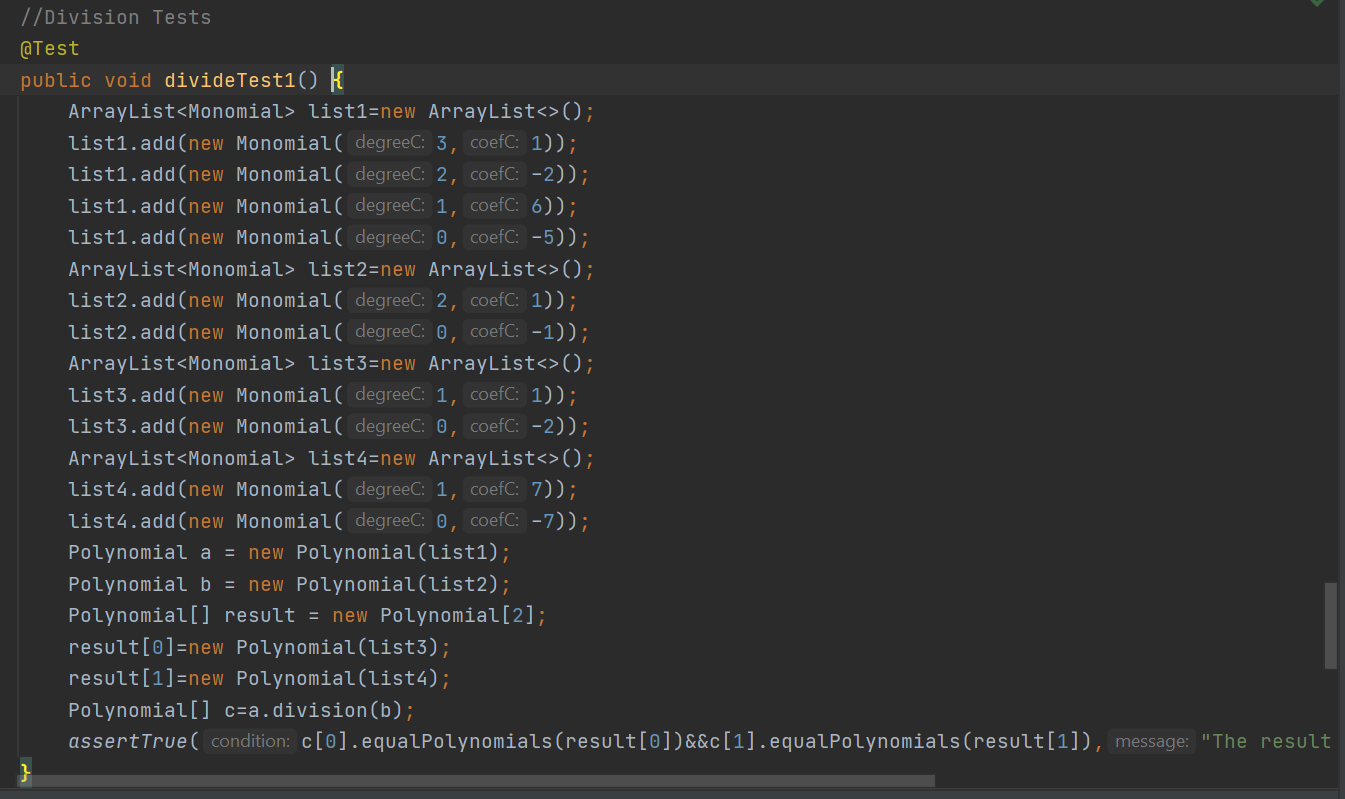




## Impartire

Polinoamele pe care s-au facut testele:

* Test 1: P(x) = 1x^3-2x^2+6x^1-5x^0 si Q(x) = 1x^2-1x^0 => C(x) = 1x^1-2x^0 si R(x) = 7x^1-7x^0
* Test 2: P(x) = 1x^2-1x^0 si Q(x) = 1x^3-2x^2+6x^1-5x^0 => C(x) = 1x^1-2x^0 si R(x) = 7x^1-7x^0
* Cazurile de testare: exemplul din prezentare; primul polinom are gradul mai mare, al doilea polinom are grad mai mare

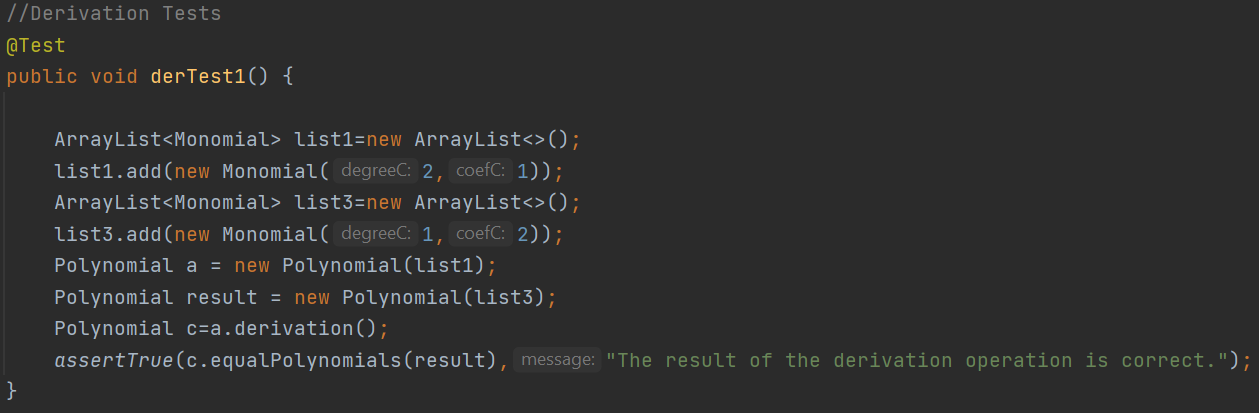


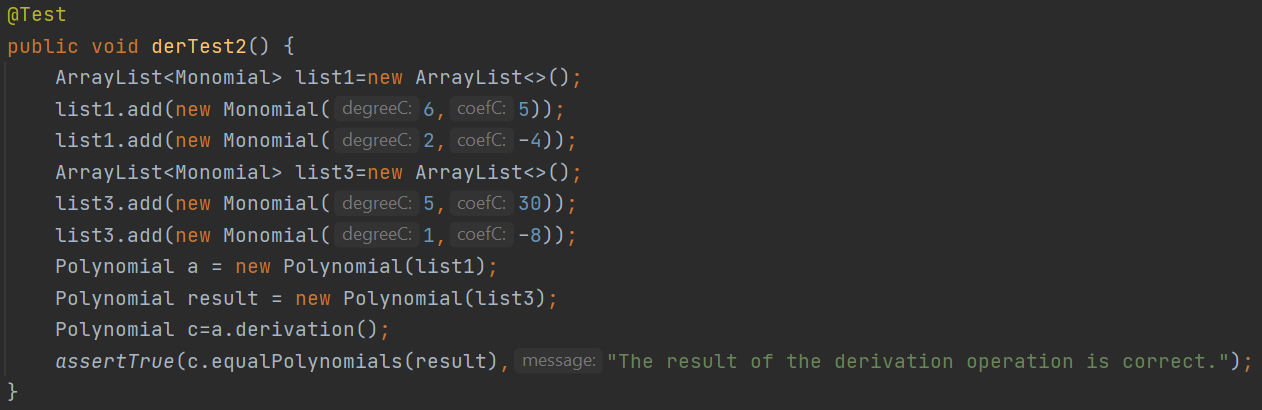


## Derivare

Polinoamele pe care s-au facut testele:

* Test 1: P(x) = x^2 => C(x) = 2x^1
* Test 2: P(x) = 5x^6-4x^2 => C(x) = 30x^5-8x^1
* Cazurile de testare: un singur monom, un polinom din mai multe monoame dintre care unul cu coeficient negativ

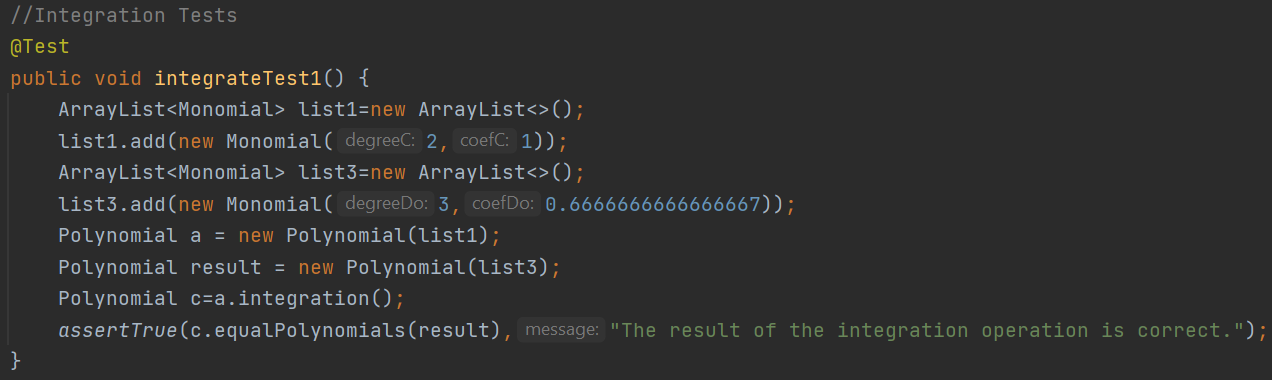


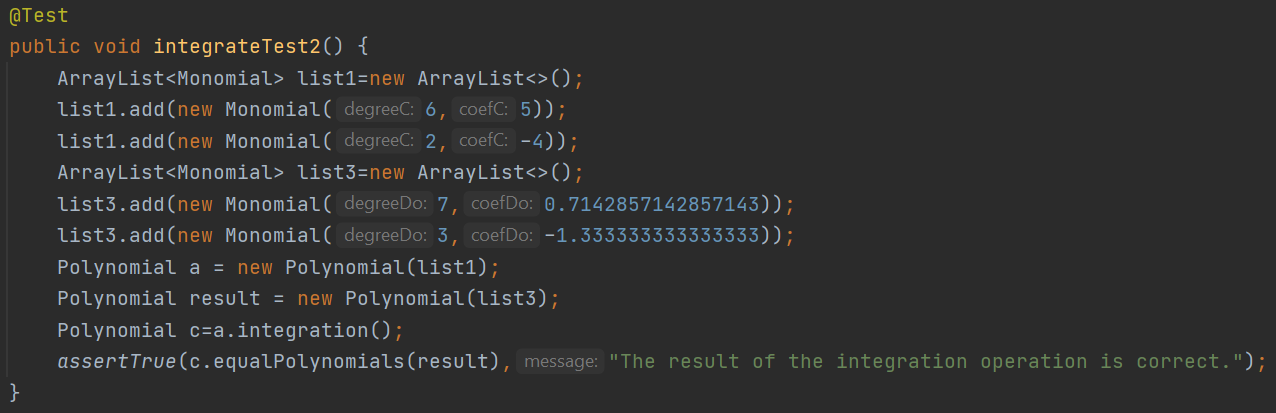


## Integrare

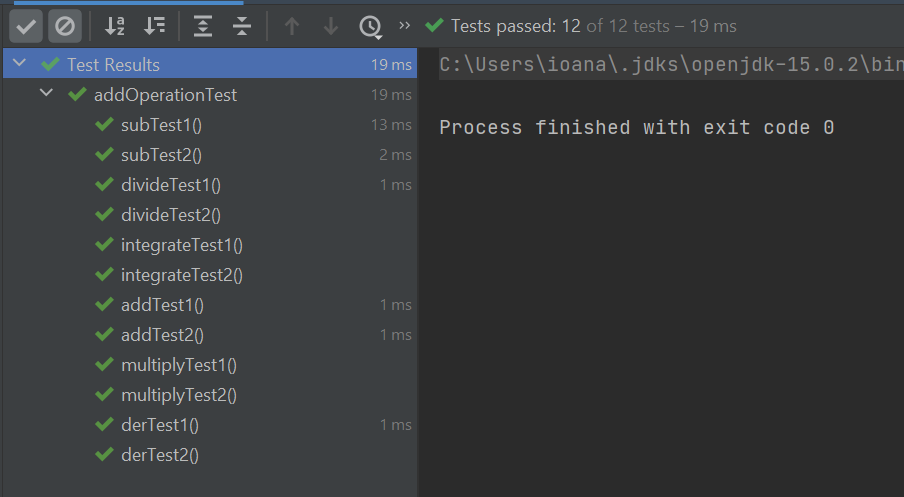
Polinoamele pe care s-au facut testele:

* Test 1: P(x) = x^2 => C(x) = 0.66x^3 ;
* Test 2: P(x) = 5x^6-4x^2 => C(x) = 0.71x^7-1.33x^3;
* Cazurile de testare: un singur monom, un polinom din mai multe monoame dintre care unul cu coeficient negativ





## Rezultate testare



# Concluzii

## Dezvoltari ulterioare

* Imbunatatirea acoperirii cazurilor speciale pentru o functionalitate mai buna
* Interfata grafica mai atractiva
* Adaugarea de o varietate mai mare de exceptii

## Ce am invatat

* Pasii care trebuie parcursi in dezvoltarea unui program
* Familiarizarea cu mediul IDE IntelliJ
* O organizare si structurare mai buna a proiectelor Java

# Bibliografie

* Indrumatorul de laborator de la POO
* Prezentarea suport oferita sub format pdf
* <https://regex101.com/>
* http://stackoverflow.com/