**CALCULATOR**

**POLINOAME**

**IRIMIA LAURA-MARIA**

**GRUPA 302210**

**PROFESOR LABORATOR: MITREA DAN**

**Cuprins:**

1. **Obiectivul temei ...............................................................................................................3**

**1.1. Obiectivul principal ....................................................................................................3**

**1.2. Obiectivele secundare ................................................................................................3**

1. **Analiza problemei, modelare, scenarii, cazuri de utilizare .................................................3**
2. **Proiectare .........................................................................................................................4**
3. **Implementare ...................................................................................................................6**
4. **Rezultate .........................................................................................................................17**
5. **Concluzii ..........................................................................................................................18**
6. **Bibliografie ......................................................................................................................18**
7. **Obiectivul temei**

**1.1. Obiectivul principal**

Obiectivul principal al acestei teme este proiectarea și implementarea unui calculator de polinoame și a unei interfețe grafice asociată acestuia prin care să se poată introduce două polinoame cu coeficienți întregi asupra cărora să se efectueze următoarele operații matematice: adunare, scădere, înmulțire, împărțire, derivare și integrare.

**1.2. Obiective secundare**

|  |  |
| --- | --- |
| **Obiectiv secundar** | **Descriere** |
| Dezvoltarea de use case-uri și scenarii | Căutarea de scenarii în care aplicația dezvoltată o să fie folosită și de către cine va fi folosită. |
| Alegerea structurilor de date | Proiectarea unor structuri eficiente și potrivite cerințelor date. |
| Împărțirea pe clase | Alegerea unor clase care vor ușura proiectarea problemei. |
| Dezvoltarea algoritmilor | Implementarea algoritmilor necesari rezolvării problemei și rezolvării cazurilor speciale. |
| Implementarea soluției | Combinarea tuturor algoritmilor implementați in vederea rezolvării problemei. |
| Testare | Găsirea cazurilor care pot crea probleme aplicației dezvoltate și testarea acestora. |

1. **Analiza problemei, modelare, scenarii, cazuri de utilizare**

Un polinom este un ansamblu de monoame care, la rândul lor, sunt formate dintr-o pereche de coeficient-exponent (putere). Un monom este exprimat în felul următor: ax^b, unde a este coeficientul și b este exponentul (puterea). Pentru ușurința în implementarea polinomului, ne-a fost recomandata folosirea unei liste de monoame (ArrayList). În interfața creată, polinomul poate fi introdus ca „ax^2+bx+c” și tot în acest fel va fi afișat și rezultatul calculului corespunzător. Identificarea coeficienților și exponenților am realizat-o folosind Regex. Cu ajutorul Pattern-ului identific câte un Match în String-ul introdus ca și input, care reprezintă un monom, iar monomul respectiv este împărțit în două grupe, prima grupă reprezentând coeficientul și a doua grupă reprezentând exponentul. Pentru o identificare corectă a coeficientului și a exponentului, eu modific String-ul introdus prin adăugarea coeficientului „1” acolo unde se găsește un „x” fără coeficient, prin adăugarea puterii „^1” acolo unde se găsește un „ax” și prin adăugarea unui „x^0” dacă în String-ul introdus avem un coeficient corespunzător lui „x” la puterea „0”.

Am încercat să fac o interfață cât mai ușor de utilizat astfel încât și o persoană nespecializată să o poată folosi cu ușurință prin adăugarea de mesaje de eroare și prin simplitatea în sine a interfeței.

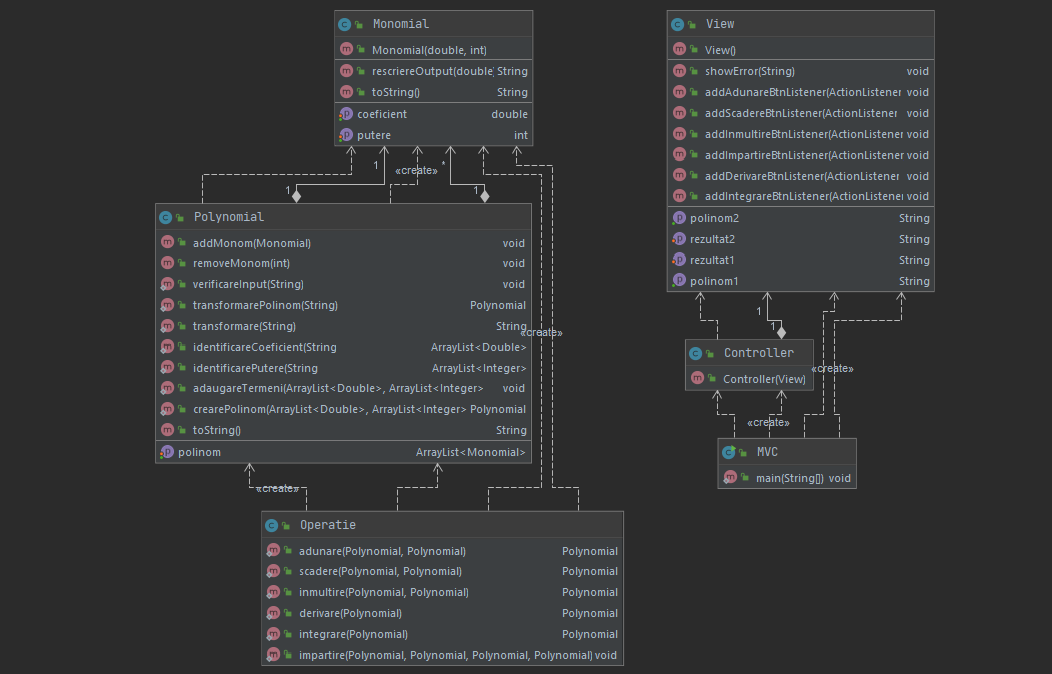
1. **Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfețe, relații, packages, algoritmi, interfața utilizator)**

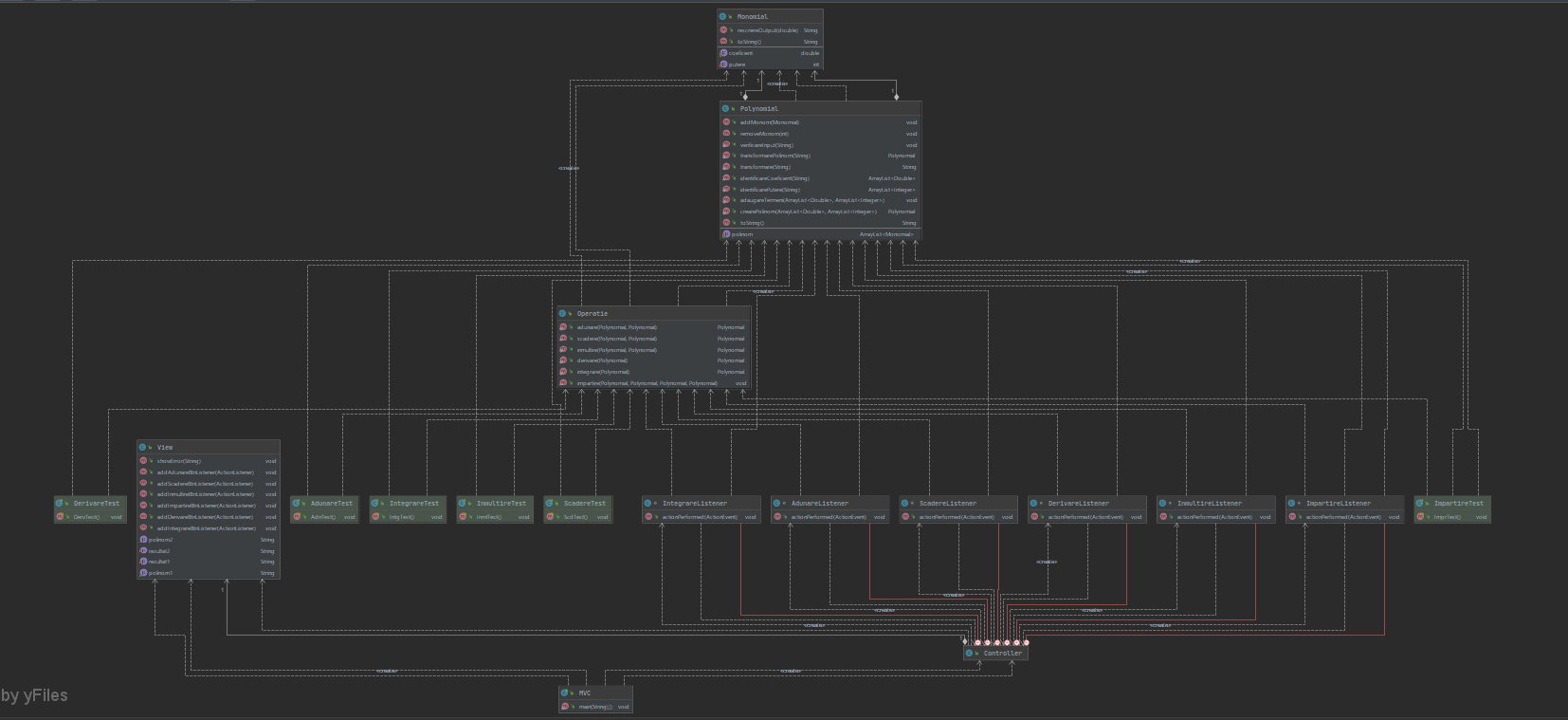
|  |  |
| --- | --- |
| **Package** | **Class** |
| main -> **java** | MVC |
| main -> java -> **model** | Monomial  Polynomial  Operatie |
| main -> java -> **controller** | Controller |
| main -> java -> **view** | View |
| test -> **java** | AdunareTest  ScadereTest  InmultireTest  ImpartireTest  DerivareTest  IntegrareTest |

|  |  |
| --- | --- |
| **Class** | **Descriere** |
| Monomial | Clasa are două atribute, coeficient de tip double și putere (exponent) de tip int, un constructor și metodele standard de set și get. Pe lângă metodele de bază, am mai adăugat o metodă care să mă ajute să fac o afișare corespunzătoare monomului și metoda suprascrisă toString(). |
| Polynomial | Clasa are un singur atribut, o lista (ArrayList) de monoame, și metodele standard de set și get și metoda suprascrisă toString(). Pe lângă aceste metode, am adăugat o metodă care adaugă un nou monom în listă și o metodă care șterge un monom de la o anumită poziție din listă. Restul metodelor din aceasta clasă sunt pentru verificarea String-ului introdus și pentru transformarea din String în polinom. |
| Operatie | Clasa nu are niciun atribut, însă conține toate metodele statice ale operațiilor implementate: adunare, scădere, înmulțire, împărțire, derivare și integrare. |
| View | Clasa are ca atribute toate componentele de la nivel de interfață care necesită manipularea ulterioară. Conține constructorul interfeței, o metodă care afișează o fereastră de eroare, metode care returnează textul din TextField-uri, metode care setează rezultatul în eticheta corespunzătoare și metode care adaugă ascultători pentru fiecare buton în parte. |
| Controller | Clasa care leagă clasele din pachetul model cu clasa View. Are un singur atribut, și anume un obiect de tip View. În cadrul constructorului inițializez atributul de tip View și asociez metodele ascultătoare pentru fiecare buton cu clasa internă corespunzătoare. |
| MVC | Clasa care conține metoda main. Aici creez un obiect de tip View pe care îl pasez constructorului Controller-ului. |
| AdunareTest | Clasa care conține testul pentru operația de adunare. |
| ScadereTest | Clasa care conține testul pentru operația de scădere. |
| InmultireTest | Clasa care conține testul pentru operația de înmulțire. |
| ImpartireTest | Clasa care conține testul pentru operația de împărțire. |
| DerivareTest | Clasa care conține testul pentru operația de derivare. |
| IntegrareTest | Clasa care conține testul pentru operația de integrare. |

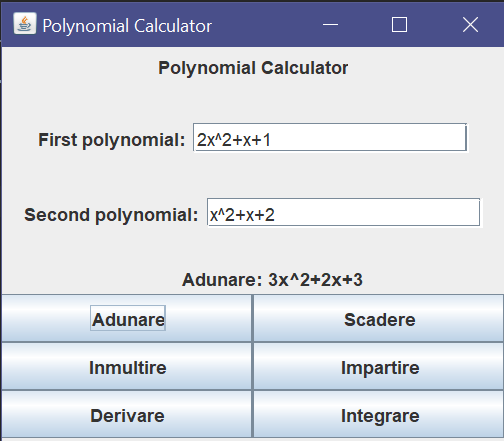
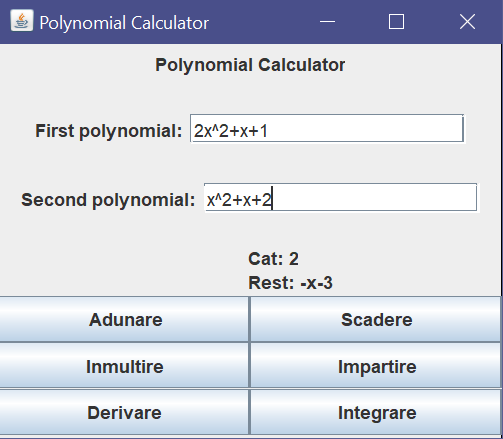
Proiectarea claselor în acest mod m-a ajutat să înțeleg mai bine legăturile dintre clase și să dezvolt algoritmii necesari în clase corespunzătoare. În pachetul „model” am pus clasele „Monomial”, „Polynomial” și „Operație”, pentru că aceste clase se ocupă de „calculele” aplicației dezvoltate. În pachetul „view” se află clasa „View” cu implementarea interfeței și metodele necesare, iar în pachetul „controller” am doar clasa „Controller” care leagă „calculele” aplicației cu interfața.

**Diagrama UML:**





**Interfața:**

**** ****

Singurul algoritm de care m-am folosit ar fi algoritmul de interclasare pe care l-am folosit pentru metodele de adunare și scădere care vor fi dezvoltate pe larg în cele ce urmează.

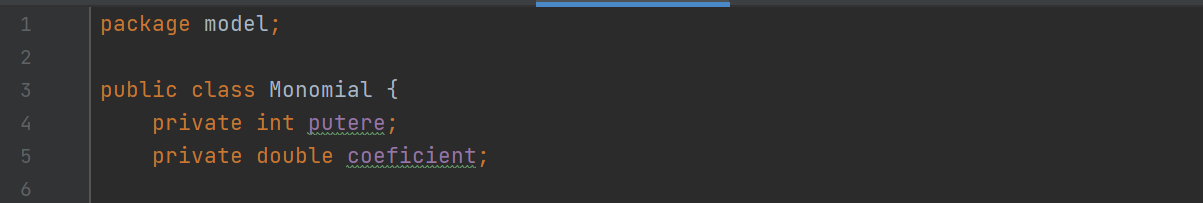
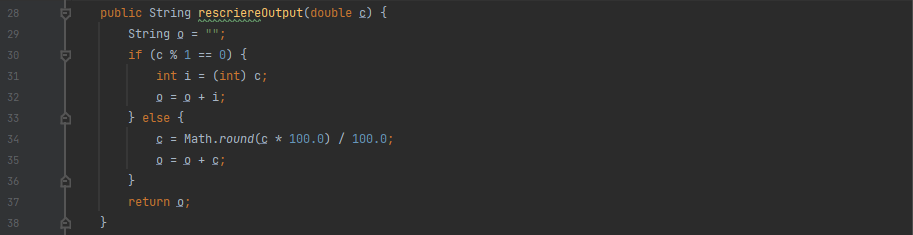
1. **Implementare**

**Clasa „Monomial”:**

Clasa „Monomial” are doar două atribute, unul de tip double care reprezintă coeficientul și unul de tip int care reprezintă puterea (exponentul) monomului. Cele mai importante metode din această clasă sunt metodele: „rescriereOutput” care returnează un String și metoda suprascrisă „toString”.

Metoda „rescriereOutput” este apelată numai în cadrul metodei „toString”. Am scris această metoda pentru a mă ajuta în afișarea unui polinom cât mai plăcut vizual și anume: String-ul returnat de această metoda reprezintă coeficientul monomului cu care este apelată metoda; dacă coeficientul este de fapt un int și nu un double, îl transformă în int și îl adaugă String-ului de returnat; dacă totuși coeficientul este double, îl rotunjește la două zecimale și îl adaugă String-ului de returnat.

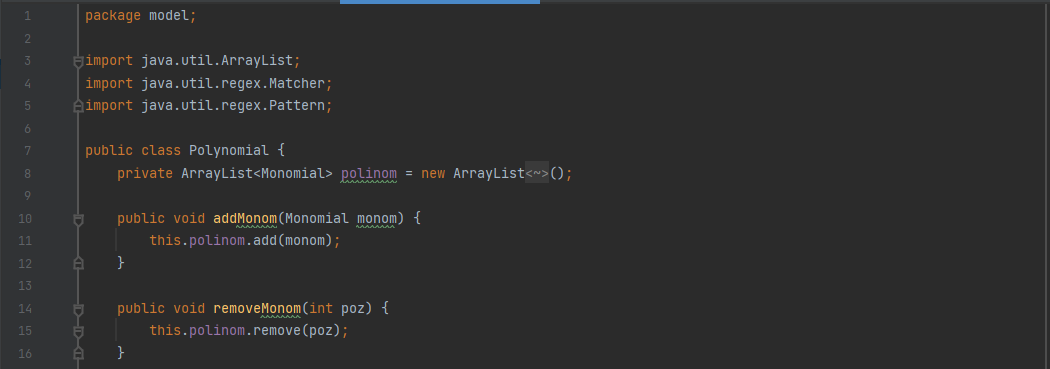
Metoda „toString” este suprascrisă în așa fel încât să trateze fiecare mod în care ar putea arăta un monom, în funcție de puterea și de coeficientul acestuia. Am luat fiecare caz în parte și m-am gândit cum ar trebuie să arate monomul afișat și mi-am creat String-ul care trebuie returnat.

****

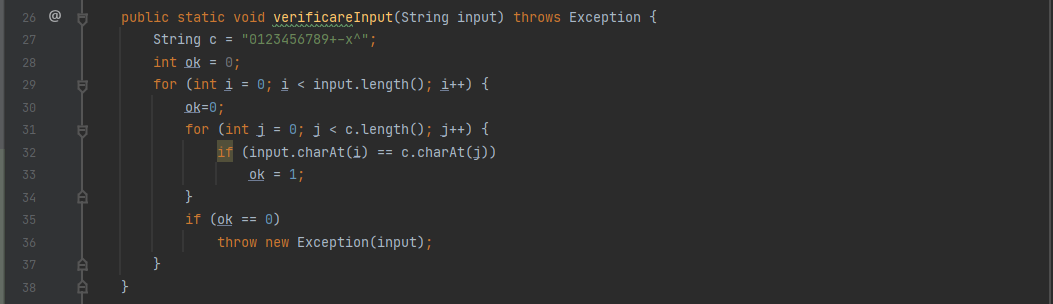


**Clasa „Polynomial”:**

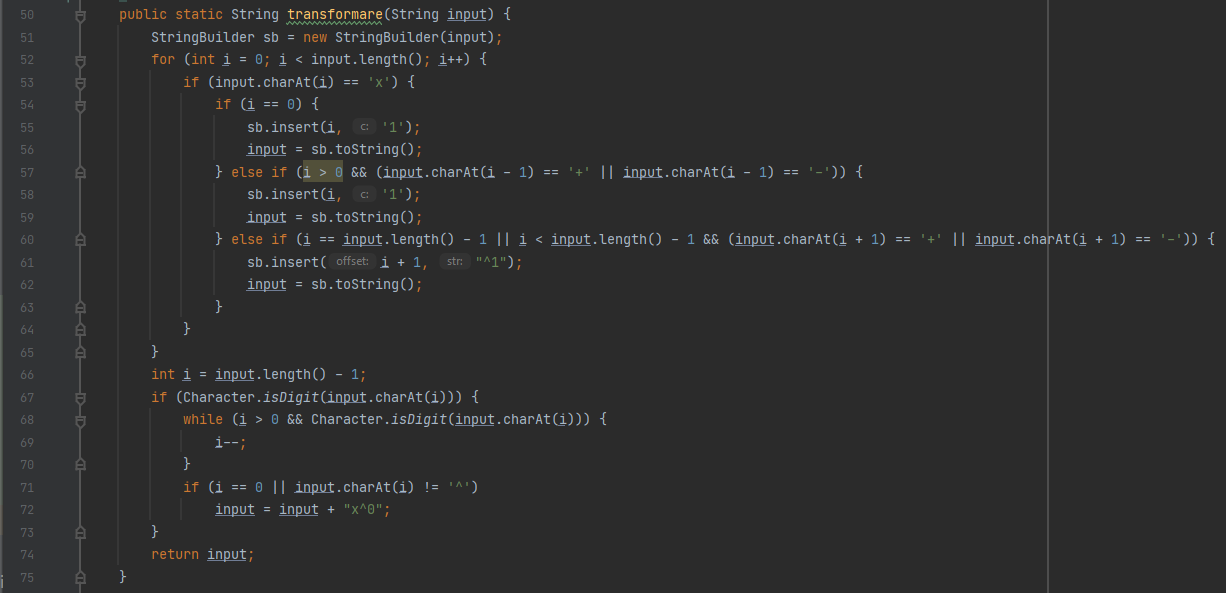
Clasa „Polynomial” are un singur atribut, „polinom”, care reprezintă lista de monoame. Mai jos este metoda care adaugă un monom la lista de monoame și care șterge un monom de pe o anumită poziție din listă.



Metoda „verificareInput” este metoda pe care o folosesc în clasa „Controller” pentru a verifica dacă ceea ce a introdus utilizatorul este asemănător cu un polinom. Această metodă testează dacă utilizatorul a introdus alte caractere, cu excepția numerelor, caracterului „+”, caracterului „-”, caracterului „^” și a literei „x”. Dacă în input-ul introdus de utilizator se găsește un caracter în plus față înșirate mai sus, metoda aruncă o excepție pe care o prind în metodele care implementează ascultătorii butoanelor.



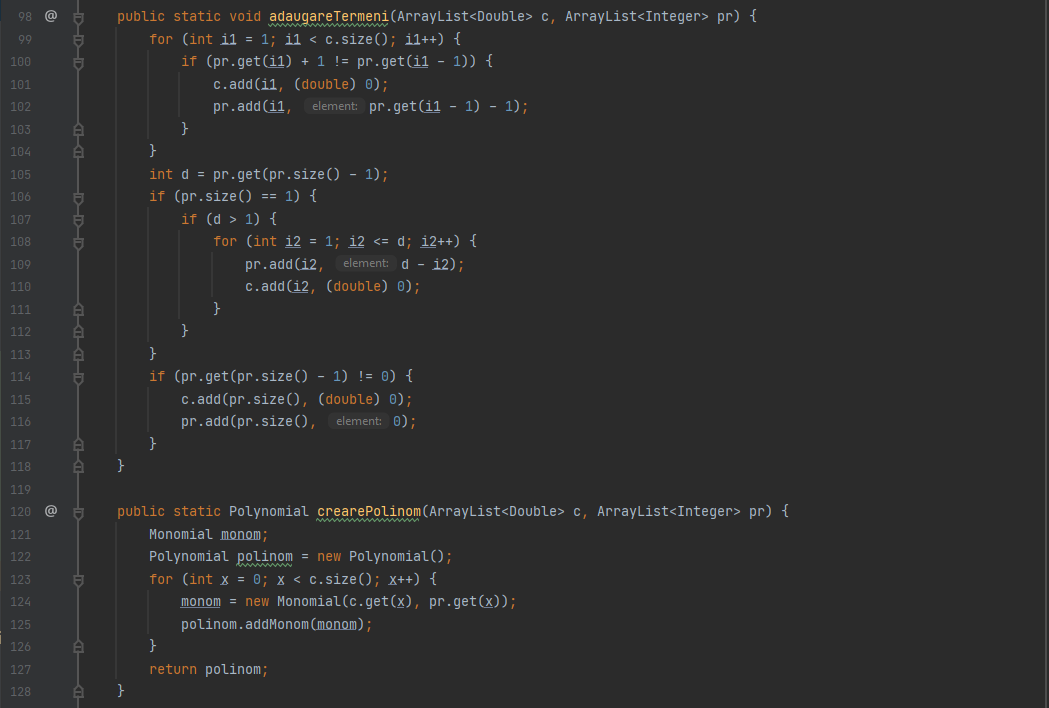
Metoda „transformare” returnează un String creat din input-ul dat de utilizator, primit ca parametru, la care am adăugat coeficienții și puterile lipsă, cum ar fi: **„1”** în fața lui „+x”, „-x” , „+x^a”, „-x^a”; **„^1”** după „+x”, „-x”, „+ax”, „-ax”; **„x^0”** după ultimul număr din polinom, dacă acesta ar trebuie să fie coeficientul lui „x” la puterea zero. Am făcut această rescriere a String-ului și inclusiv această metodă pentru a-mi fi mai ușor să identific coeficientul și puterea monoamelor.



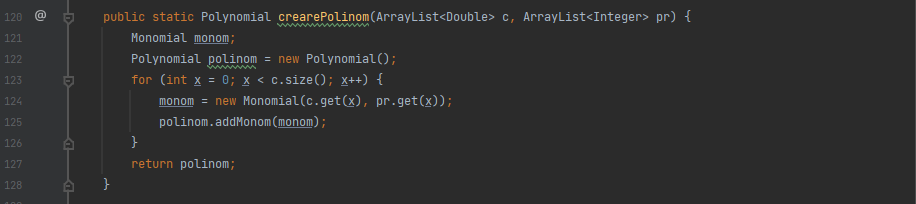
Metodele „identificareCoeficient” și „identificarePutere” primesc ca parametru String-ul care a trecut prin metoda „transformare” de mai sus și găsesc Match-urile pe baza Pattern-ului creat de mine. În prima metoda identific coeficienții, care reprezintă primul grup din Pattern, iar în a doua metoda identific puterile, care reprezintă al doilea grup. Ambele metode returnează două liste, una o listă de double și cealaltă o listă de int.



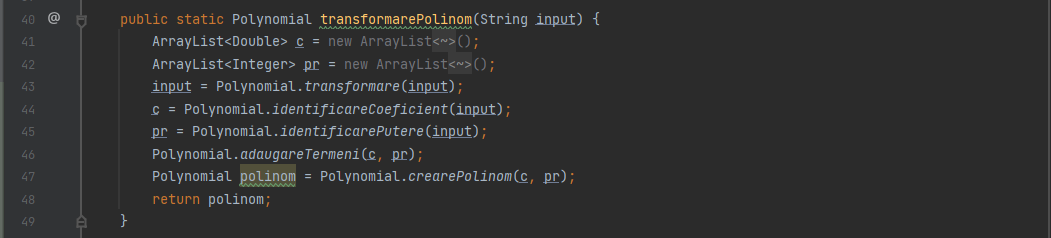
Metoda „adaugareTermeni” primește ca parametru cele două liste returnate mai sus și adaugă în ele coeficienții și puterile lipsă. Am gândit operațiile pe baza calculelor cu polinoame pe care le pot face în Matlab. Dacă aș avea un polinom de genul: „x^3+x” cu lista de coeficienți egală cu [1 1] și lista de puteri egală cu [3 1], această metodă transformă lista de coeficienți în [1 0 1 0] și lista de puteri în [3 2 1 0]. Practic, metoda adaugă puterile în ordine descrescătoare până la 0, inclusiv 0 și adaugă în lista de coeficienți 0 în dreptul puterilor care nu se găsesc în polinom.



Metoda „crearePolinom” primește ca parametrii listele modificate în metoda „adaugareTermeni” și creează pentru fiecare pereche coeficient-putere monomul aferent pe care îl adaugă la polinomul ce va fi returnat de această metodă.



Metoda „transformarePolinom” am creat-o pentru a nu avea cod duplicat. În această metodă se trece prin toți pașii descriși în metodele de mai sus pentru returnarea polinomului creat din String-ul input introdus de utilizator.



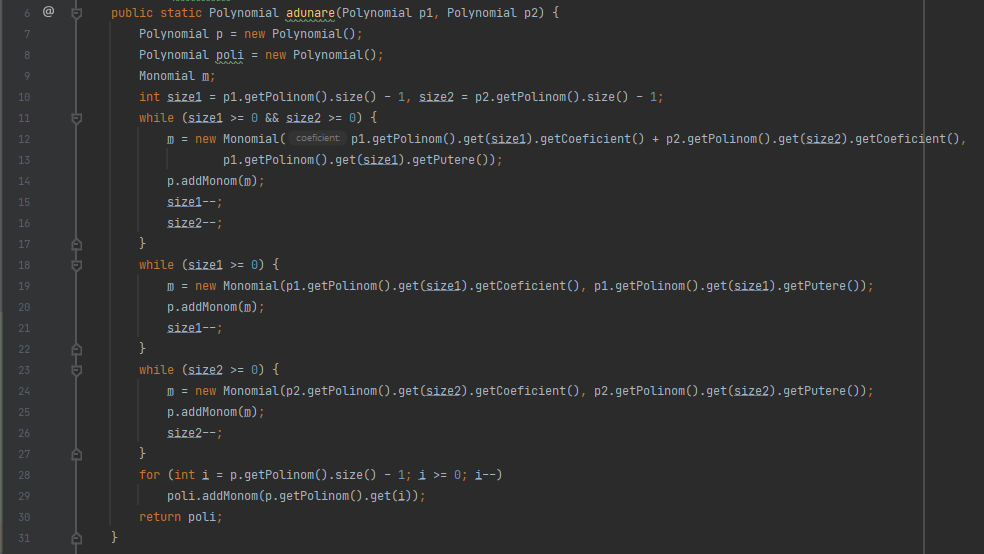
Metoda suprascrisă „toString” se folosește de metoda „toString” din clasa „Monomial”, însă mai are și câteva aspecte în plus. Dacă toți coeficienții monoamelor din polinomul pe care vreau să îl afișez sunt zero, atunci String-ul returnat va fi „0”. Dacă nu toți coeficienții sunt zero, atunci pentru fiecare String aferent monomului corespunzător, dacă acesta este diferit de „null” (metoda „toString” din clasa „Monomial” returnează „null” dacă coeficientul este zero), atunci acesta este adăugat la String-ul polinomului. Ultimul „if” testează dacă pe prima poziție din String-ul de returnat s-ar afla semnul „+”; dacă găsește semnul „+” acolo, atunci îl elimină.



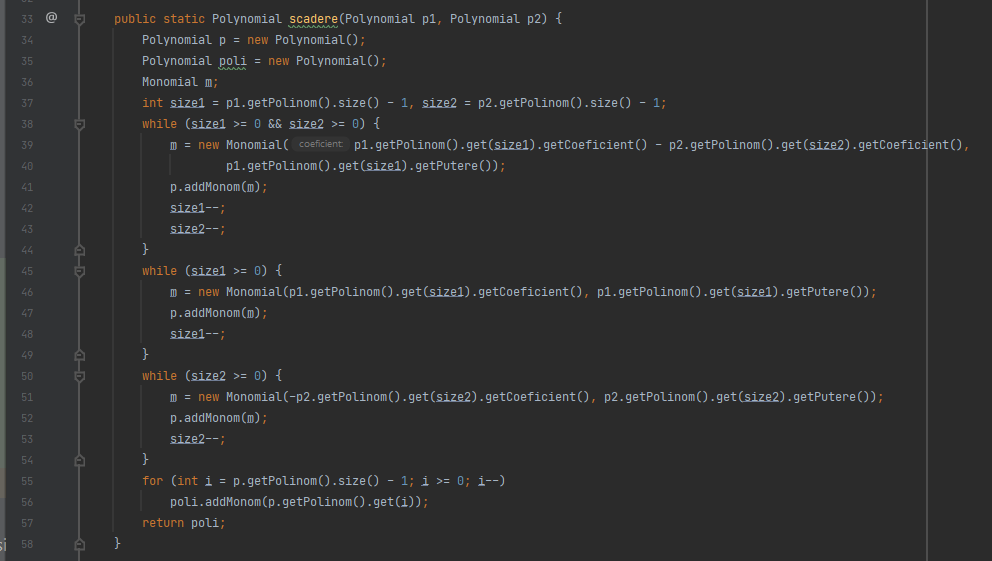
**Clasa „Operatie”:**

Clasa „Operație” nu are niciun atribut, însă această conține toate metodele pentru operații, implementate static.

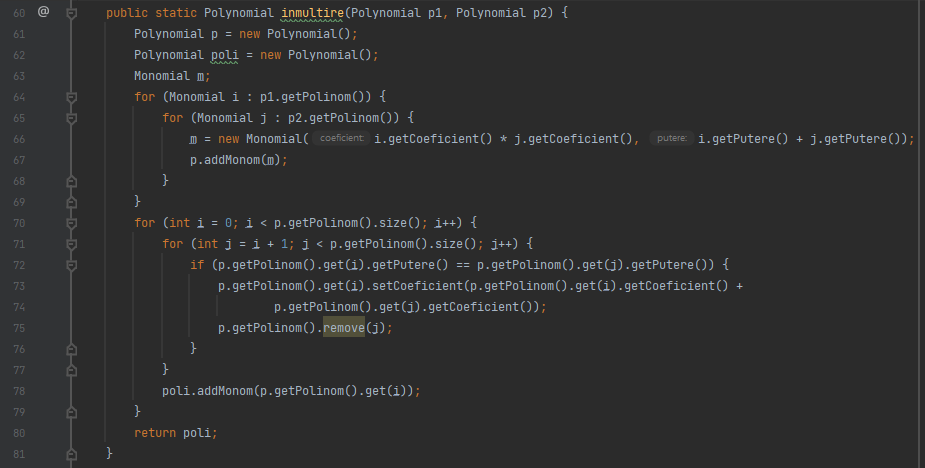
Metoda „Adunare” returnează un polinom, care reprezintă suma polinoamelor primite ca parametru. Pentru implementarea acestei metode m-am ajutat de algoritmul de interclasare. Mi-am creat două variabile în care am salvat dimensiunea polinoamelor primite ca parametru. În primul „while” parcurg fiecare polinom de la final, în așa fel asigurându-mă că adun coeficienții de la aceleași puteri. Îmi creez un nou monom care primește ca și coeficient suma dintre coeficienții celor două polinoame și ca și putere, puterea unuia dintre polinoame (puterile sunt egale având în vedere că parcurg polinoamele începând de la cea mai mică putere la cea mai mare). Monomul creat este adăugat la polinomul „p”, iar variabilele în care am salvat dimensiunea polinoamelor scad cu câte o unitate. În următoarele două „while”-uri verific dacă mai am elemente de adăugat din cele două polinoame. Având în vedere că primul „while” se oprește atunci când am terminat de parcurs unul dintre polinoame, trebuie să verific în care dintre ele mai sunt elemente care ar trebui adăugate la sumă. Dacă mai găsesc elemente care ar trebui adăugate, le adaug. Ultimul „for” îl folosesc ca să întorc polinomul „p”, cel in care am salvat suma polinoamelor până acum, și îl salvez în polinomul „poli” pe care îl returnez. Am făcut această întoarcere pentru că în polinomul „p”, monoamele au fost salvate în ordinea crescătoare a puterilor și eu vreau să afișez polinomul în ordinea descrescătoare a puterilor monoamelor.

****

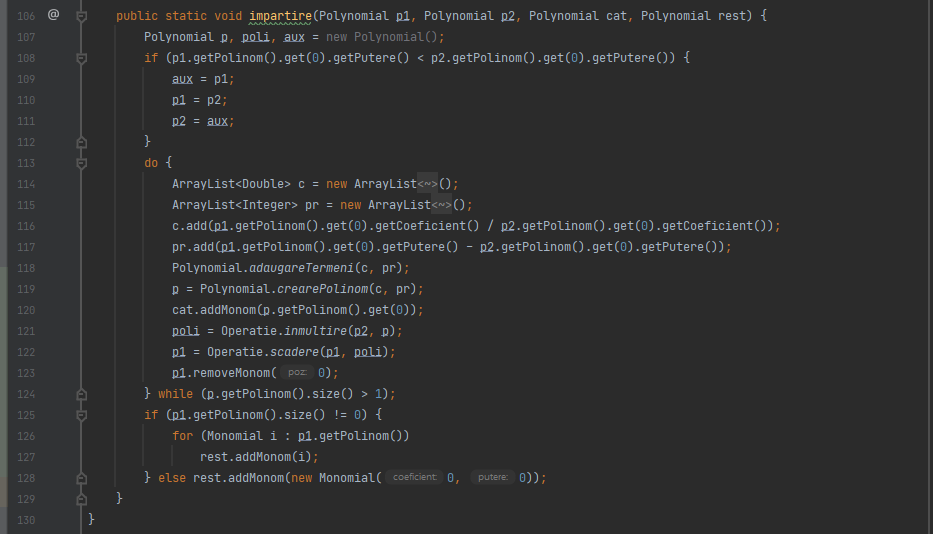
Metoda „scadere” este la fel cu metoda „adunare”, numai că în locul adunării coeficienților celor două polinoame în monomul creat, acum scad coeficienții.

****

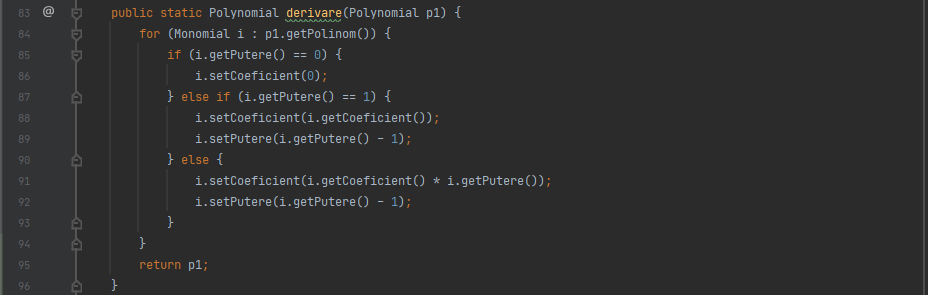
Metoda „inmultire” efectuează înmulțirea polinoamelor primite ca parametru. În primele două „for”-uri îmi creez un polinom, „p”, care conține fiecare monom care primește ca și coeficient înmulțirea coeficienților celor două monoame aferente din polinoame și ca și putere, suma puterilor celor două monoame. După terminarea acestor două „for”-uri voi avea un polinom care nu conține puteri unice, lucrul pe care îl rezolv în următoarele două „for”-uri. Parcurg polinomul creat până acum și dacă găsesc pe o poziție următoare decât cea pe care mă aflu la momentul actual două puteri egale, atunci adun coeficienții celor două monoame cu puteri egale găsite și șterg monomul de pe poziția găsită. După ce ies din „for”-ul cu j, adaug la polinomul „poli”, care va fi și returnat, monomul de pe poziția „i”.



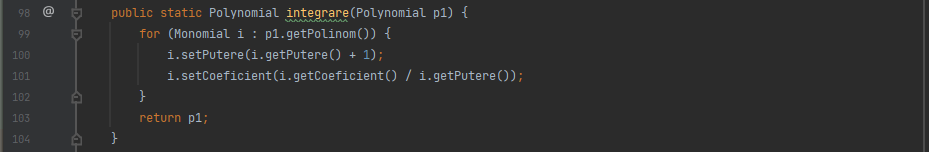
Metoda „impartire” efectuează împărțirea celor două polinoame primite ca și parametru și salvează câtul și restul în polinoamele aferente primite tot ca parametru. Mai întâi mă asigur că primul polinom are gradul mai mare decât al doilea; dacă nu e așa, le interschimb prin metoda celor trei pahare. Apoi, conform calculelor pe care le-am efectuat în prealabil cu pixul pe foaie, îmi creez două liste pentru coeficienți și puteri. În lista pentru coeficienți adaug împărțirea dintre coeficientul primului monom al primul polinom cu coeficientul primului monom al celui de-al doilea polinom, iar în lista de puteri adaug puterea primul monom al primului polinom minus puterea primului monom al celui de-al doilea polinom. Apoi urmez pașii necesari descriși mai sus pentru a-mi crea un polinom cu listele create. În „cat” adaug primul monom din polinomul creat. Apoi înmulțesc polinomul creat cu primul polinom, cel cu puterea cea mai mare și salvez rezultatul în „poli”. Apoi suprascriu primul polinom cu diferența dintre „p”, polinomul creat cu cele două liste, și „poli”, polinomul creat din înmulțirea primului polinom cu „p”. Apoi șterg primul monom din primul polinom pentru că acesta are coeficientul 0, conform principiului împărțirii polinoamelor. Fac acești pași cât timp polinomul creat „p” are dimensiunea mai mare decât 1 (atât timp cât nu am ajuns la finalul împărțirii, adică nu am ajuns la x la puterea 0, adică un număr simplu). După ce am creat câtul, urmează să creez și restul. Pentru acest lucru, dacă în primul polinom mai avem monoame, atunci aceste monoame sunt adăugate restului; dacă nu, atunci restul este 0.



Metoda „derivare” efectuează operația de derivare a polinomului primit ca și parametru, pe care îl și returnează. Pentru fiecare monom din polinomul primit, dacă avem un număr simplu, atunci acesta dispare din polinom (îi setez coeficientul pe 0); dacă avem un „x”, atunci coeficientul monomului rămâne neschimbat și puterea acestuia scade; dacă puterea este mai mare decât 1, atunci coeficientul monomului devine înmulțirea dintre coeficientul curent și puterea curentă, iar puterea acestuia scade.



Metoda „integrare” efectuează operația de integrare a polinomului primit ca și parametru, pe care îl și returnează. Pentru fiecare monom din acest polinom, îi cresc puterea și îi setez coeficientul ca raportul dintre coeficientul curent și puterea mărită.



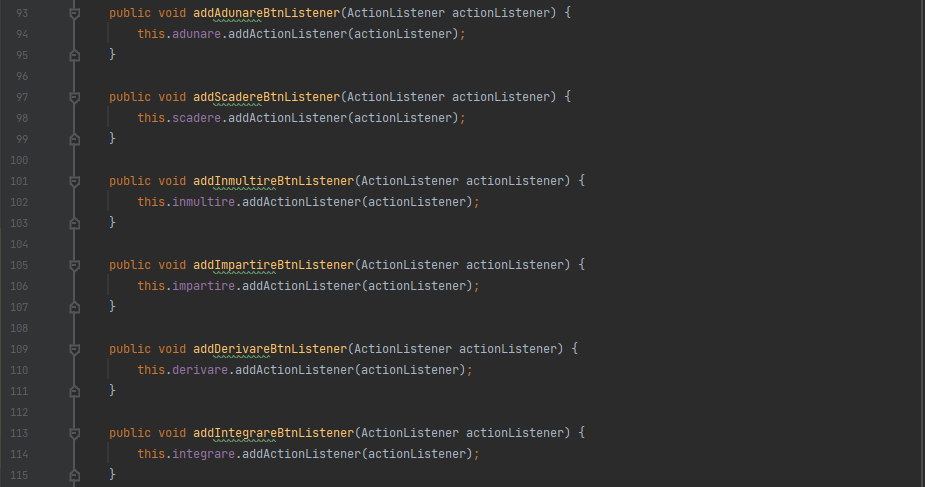
**Clasa „View”:**

Clasa „View” care extinde „JFrame” are ca atribute cele mai importante componente din cadrul interfeței (unele componente au fost declarate numai în constructor). În cadrul constructorului este creată fereastra cu anumite dimensiuni și un anumit nume, apoi este creat „Panel”-ul de bază la care se adaugă treptat restul „Panel”-urilor care conțin componentele necesare.

Metodele de mai jos din clasa „View” sunt folosite pentru a afișa o fereastră cu un mesaj de eroare, pentru a extrage input-ul adăugat de utilizator și pentru a suprascrie etichetele rezultatelor calculelor.

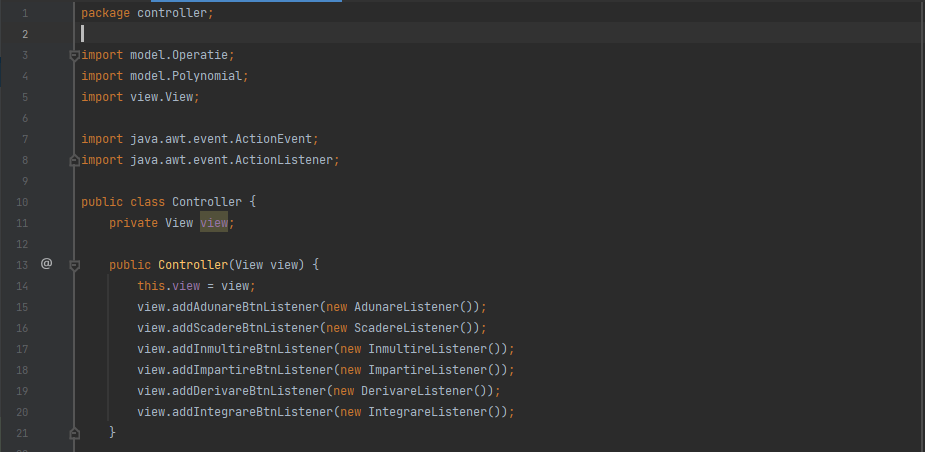


Metodele de mai jos sunt metodele care adaugă ascultători fiecărui buton în parte.



**Clasa „Controller”:**

Clasa „Controller” are un singur atribut, și anume o instanță de obiect către clasa „View”. În cadrul constructorului clasei „Controller” se inițializează singurul atribut și se implementează ascultătorii. Voi exemplifica mai jos doar una dintre cele șase metode, cea pentru adunare, pentru că toate urmează același principiu.



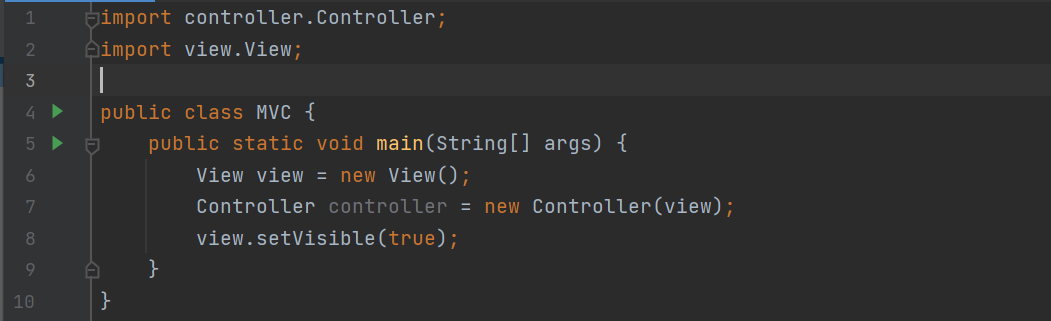
Clasa internă „AdunareListener” care implementează metoda „actionPerformed” pentru butonul de „Adunare” funcționează după principiul următor: în blocul de „try” se încearcă retragerea input-urilor date de către utilizator, care sunt mai apoi verificate dacă sunt adecvate cu metoda mai sus explicată, „verificareInput”, care aruncă o excepție care este prinsă în blocul „catch”, unde se afișează fereastra de eroare; String-urile tocmai verificate se pasează metodei „transformarePolinom” care creează cele două polinoame care vor fi pasate metodei „adunare”; rezultatul adunării este salvat în polinomul „suma” care este mai apoi afișat în una dintre cele două etichete pentru rezultat.

Pe baza acestui principiu funcționează fiecare clasă internă pentru fiecare buton.



**Clasa „MVC”:**

În clasa „MVC” creez o instanță a unui obiect de clasă „View” care este pasat ca argument constructorului „controller”-ului și apoi setez vizibilitatea „view”-ului.

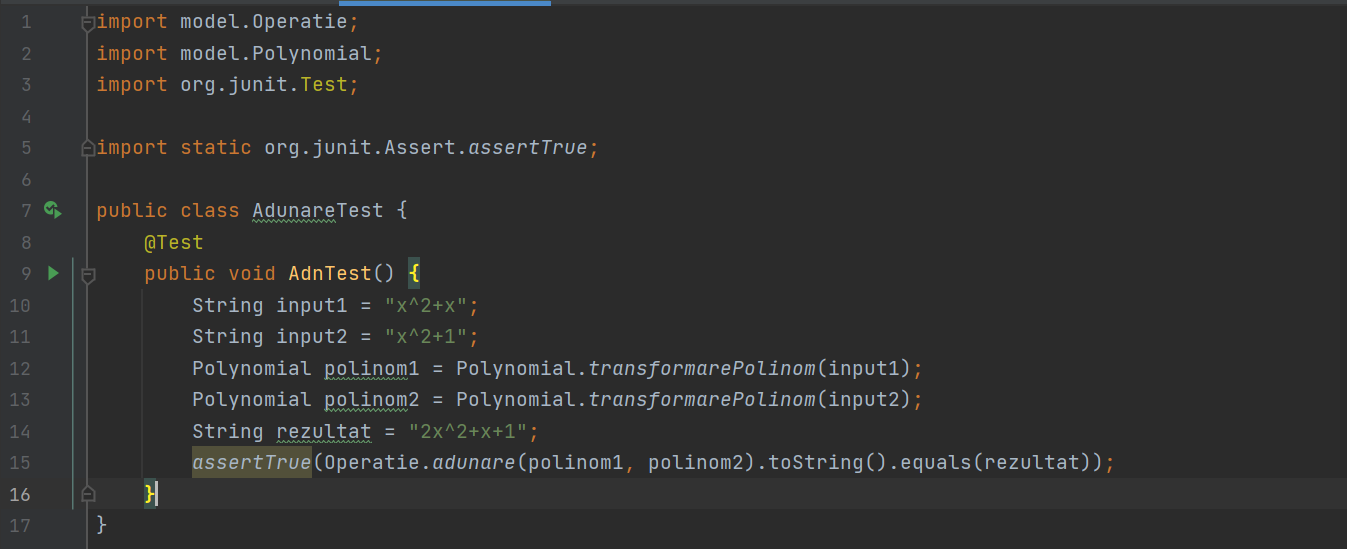


1. **Rezultate (testare)**

Am folosit JUnit pentru testarea fiecărei operații în parte. Mai jos am adăugat doar testul pentru operația de adunare pentru că restul funcționează pe același principiu.

Mi-am creat clasa „AdunareTest” unde am creat metoda de tip „Test”, „AdnTest”. În această metodă mi-am creat două String-uri din care am făcut mai apoi două polinoame. Am ales aceste două polinoame pentru că am considerat că pot crea probleme la afișare, având dimensiuni diferite. Mi-am creat încă un String unde am pus rezultatul așteptat în urma operației de adunare și mai apoi am testat dacă String-ul polinomului rezultat în urma operației de adunare este egal cu rezultatul așteptat.

Testul a trecut cu succes.



1. **Concluzii**

În urma rezolvării acestei teme, pot spune că mi-am însușit mai bine cunoștințele din semestrul întâi despre Java și am acumulat și informații noi, cum ar fi utilizarea Regex-ului. Dezvoltarea treptată a aplicației m-a ajutat să vin cu idei cât mai creative și eficiente pentru fiecare problemă care apărea.

Ca și dezvoltări ulterioare, aș mai finisa metoda care verifică dacă input-ul este cu adevărat polinom și poate aș face și o interfață mai plăcută vizual, nu doar minimalistă.

1. **Bibliografie**

Nu m-am ajutat de nicio sursă de inspirație pentru redactarea documentație. Tot ce am scris aici sunt ideile care mi-au venit pe moment. Nu pot garanta că în aceste idei nu se găsesc rezumate a altor idei citite pe diverse site-uri de domeniu sau în diverse cărți.