DOCUMENTATIE

TEMA *1*

**Polynomial Calculator**

NUME STUDENT: MOLNÁR ZSANETT-INGRID

GRUPA: 302210

# CUPRINS

[1. Obiectivul temei 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297886)

[3. Proiectare 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297887)

[4. Implementare 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297888)

[5. Rezultate 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297889)

[6. Concluzii 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297890)

[7. Bibliografie 3](file:///E:\Facultate\Anul%202\SEM%202\TP\PT2021-2022_Documentation_Template.doc#_Toc95297891)

# Obiectivul temei

Principalul obiectiv al lucrării este realizarea unui calculator polinomial cu o interfață grafică dedicată utilizatorului, prin care acesta poate insera polinomi, selecta operația matematică care dorește să se efectueze (adunare, scădere, înmulțire, împărțire, derivare, integrare), și vizualiza rezultatul obținut (prin afișarea pe ecran). Pentru a îndeplini obiectivul principal, trebuie urmați câțiva pași esențiali, și anume obiectivele secundare:

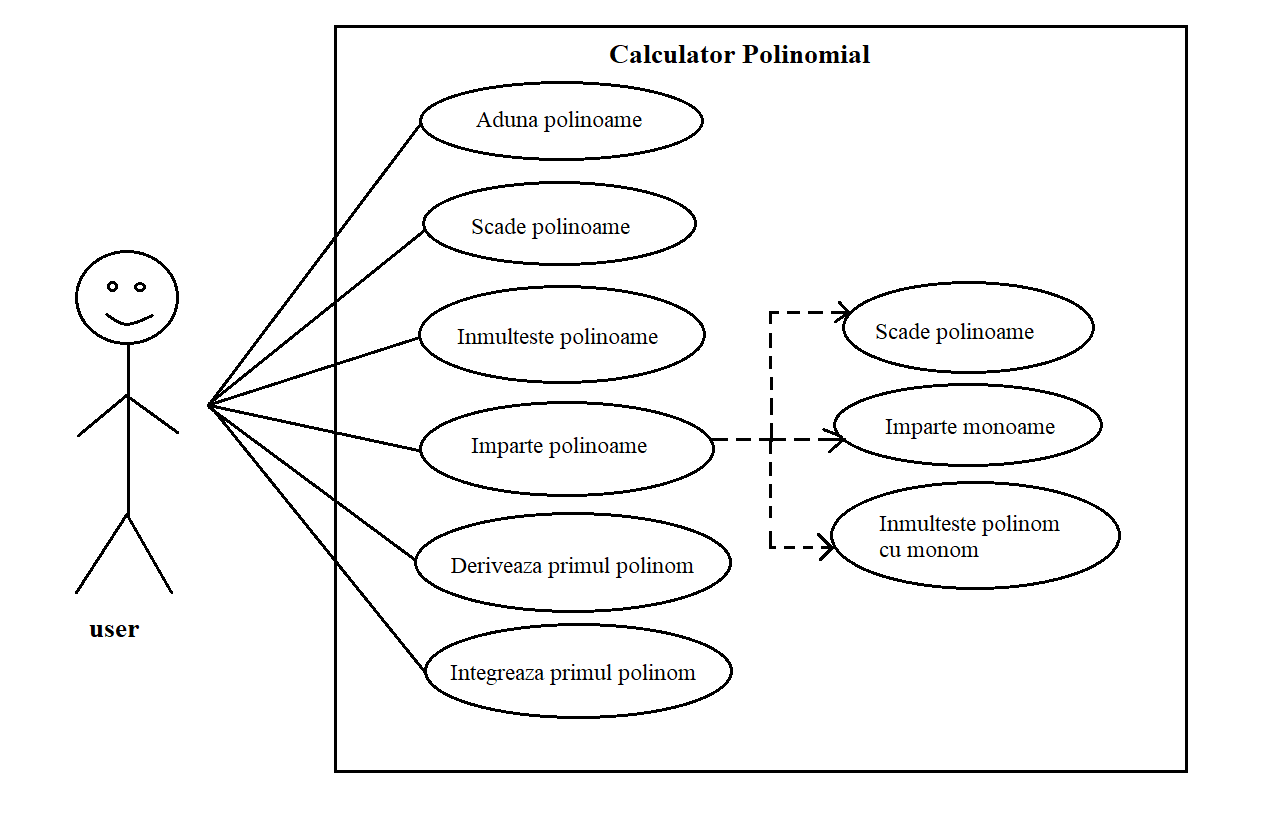
* **Analiza problemei și identificare cerințelor**: constă în identificarea principalelor obiecte și atributelor acestora (se va detialia în capitolul: *2*. *Analiza problemei, modelare, scenarii, cazuri de utilizare*)
* **Proiectarea calculatorului polinomial**: se referă la structura, arhitectura proiectului (se va detalia în capitolul: *3. Proiectare*)
* **Implementarea calculatorului polinomial:** se referă la clasele și metodele utilizate pentru realizarea operațiilor matematice dintre polinoame și crearea unei interfațe user-friendly (se va detalia în capitolul: *4. Implementare)*
* **Testarea calculatorului polinomial:** constă în testarea funcționalității proiectului folosind frameworkul JUnit (scenariile pentru testare vor fi prezentate în capitolul: *5. Concluzii)*

# Analiza problemei, modelare, scenarii, cazuri de utilizare

Cerințele funcționale ale proiectului sunt:

* Permiterea introducerii polinoamelor de către utilizator
* Permiterea selectării operației matematice de către utilizator
* Calculatorul polinomial trebuie să adune două polinoame
* Calculatorul polinomial trebuie să scadă polinoame
* Calculatorul polinomial trebuie să înmulțească două polinoame
* Calculatorul polinomial trebuie să împărțească două polinoame
* Calculatorul polinomial trebuie să deriveze un polinom
* Calculatorul polinomial trebuie să integreze un polinom

Use-caseurile sunt niște acțiuni sau evenimente care definesc interacțiunile dintre un rol și un sistem pentru a îndeplini o cerință. Aceasta diagramă use-case reprezintă use-caseurile și actorii proiectului (în cazul nostru un singur actor există, și anume userul calculatorului), respectiv relația dintre acestea.



După cum se observă, un use-case poate depinde și alte use-caseuri. În cazul nostru, operația de împărțire polinoame implică și scăderea, înmulțirea cu monom, și, desigur, împărțirea acestora.

Descrierea use-caseurilor:

* **Adunarea polinoamelor:**

Actorul principal: user

Principalul scenariu de succes:

1) Utilizatorul introduce cele două polinoame în interfața grafică a utilizatorului

2) Utilizatorul selectează operația de „adunare”, apăsând butonul “**+**”

3) Calculatorul efectuează adunarea polinoamelor și afișează rezultatul pe ecran, mai exact în căsuța rezultatului

Secvență alternativă: nu se introduc polinoamele în formatul corect

* **Scăderea polinoamelor:**

Actorul principal: user

Principalul scenariu de succes:

1) Utilizatorul introduce cele două polinoame în interfața grafică a utilizatorului

2) Utilizatorul selectează operația de „scădere”, apăsând butonul “**-**”

3) Calculatorul efectuează scăderea polinoamelor și afișează rezultatul pe ecran, mai exact în căsuța rezultatului

Secvență alternativă: nu se introduc polinoamele în formatul corect

* **Înmulțirea polinoamelor:**

Actorul principal: user

Principalul scenariu de succes:

1) Utilizatorul introduce cele două polinoame în interfața grafică a utilizatorului

2) Utilizatorul selectează operația de „înmulțire”, apăsând butonul “×”

3) Calculatorul efectuează înmulțirea polinoamelor și afișează rezultatul pe ecran, mai exact în căsuța rezultatului

Secvență alternativă: nu se introduc polinoamele în formatul corect

* **Împărțirea polinoamelor:**

Actorul principal: user

Principalul scenariu de succes:

1) Utilizatorul introduce cele două polinoame în interfața grafică a utilizatorului

2) Utilizatorul selectează operația de „împărțire”, apăsând butonul “**/**”

3) Calculatorul efectuează împărțirea polinoamelor și afișează rezultatul (câtul și restul) pe ecran, mai exact în căsuțele rezultatului

Secvență alternativă: nu se introduc polinoamele în formatul corect

* **Derivarea unui polinom:**

Actorul principal: user

Principalul scenariu de succes:

1) Utilizatorul introduce un polinom în interfața grafică a utilizatorului

2) Utilizatorul selectează operația de „derivare”, apăsând butonul “ **’** ”

3) Calculatorul efectuează derivarea polinomului și afișează rezultatul pe ecran, mai exact în căsuța rezultatului

Secvență alternativă: nu se introduce polinomul în formatul corect

* **Integrarea unui polinom:**

Actorul principal: user

Principalul scenariu de succes:

1) Utilizatorul introduce un polinom în interfața grafică a utilizatorului

2) Utilizatorul selectează operația de „integrare”, apăsând butonul “ ∫”

3) Calculatorul efectuează integrarea polinomului și afișează rezultatul pe ecran, mai exact în căsuța rezultatului

Secvență alternativă: nu se introduce polinomul în formatul corect

# Proiectare

În ceea ce privește proiectarea temei, se poate identifica următoarea schemă:

Calculator

polinomial

Rezultat

Polinom1

Polinom2

Operatie

# 

Pentru a realiza funcționalitățile calculatorului, avem nevoie de 3 intrări: 2 polinoame pe care se vor efectua o operație matematică, și operațiaîn sine, care poate fi: adunare, scădere, înmulțire, împărțire, derivare, integrare. Dacă dorim să se efectueze o operație de derivare sau integrare pe un polinom, atunci al doilea polinom nu mai e necesar (Polinom 2).

Modelele arhitecturale definesc structuri pentru sisteme software în termeni de subsisteme predefinite și responsabilităților acestora.

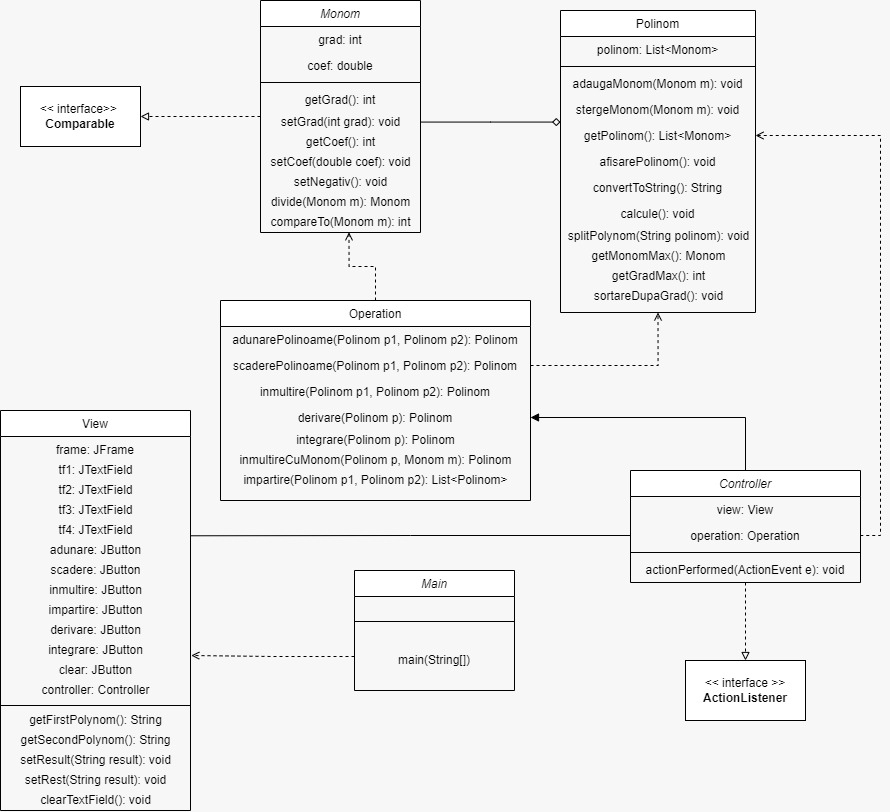
Pentru proiectarea calculatorului polinomial am ales modelul arhitectural de implementare **MVC (Model View Controller)**, care împarte aplicația noastră în trei părți:

* **Componentele Model:** încapsulează datele de bază și funcționalitățile
* **Componentele View:** afișează informații utilizatorului, obține datele pe care le afișează de la Model
* **Controller:** Fiecărui View îi este asociată o componentă Controller. Controllerele primesc intrarea de obicei sub forma unor evenimente care denotă mișcarea mouse-ului, activarea butoanelor mouse-ului sau intrarea de la tastatură. Evenimentele sunt traduse în cereri de service, care sunt trimise fie la Model sau la View.

Clasele pe care le-am utilizat pentru realizarea acestei structuri sunt următoarele:

* Clasa **Monom:** un monomreprezintă un element dintr-un polinom (ex. 2X^3). Această clasă conține atributele și metodele caracteristice unui monom.
* Clasa **Polinom:** un polinom este alcătuit din mai multe monoame. Clasa conține atributele și metodele caracteristice unui polinom.
* Clasa **Operation:** conține operațiile matematice care se pot efectua asupra polinoamelor. În modelul MVC, această clasă reprezintă clasa Model.
* Clasa **View:** în această clasă este realizată interfața utilizatorului (GUI). În modelul MVC, aceasta reprezintă clasa View.
* Clasa **Controller:** această clasă “controlează” acțiunile calculatorului, adică în cazul apăsării unui buton se ocupă de evenimentele, acțiunile care trebuie să urmeze pentru ca calculatorul nostru să îndeplinească cerințele și să funcționeze corect. În arhitectura MVC, această clasă reprezintă Controller-ul.
* Clasa **Tests**: conține testele, scenariile de testare a proiectului.
* Clasa **MainClass**: în această clasă se apelează, se deschide calculatorul polinomial.

Diagrama UML de clase și de pachete este următoarea:



# 4. Implementare

Pentru înțelegerea corectă și bună a funcționalității Calculatorului Polinomial, în acestă secțiune se vor descrie detaliat clasele menționate în secțiunile anterioare.

* Clasa **Monom**:

Această clasă implementează obiectul Monom, care reprezintă un element dintr-un polinom (ex. 3X^4). Clasa are următoarele câmpuri:

* **int grad:** reprezintă gradul monomului (ex. 3X^4 => grad=4)
* **double coef:** reprezintă coeficientul monomului (ex. 3X^4 => coef=4)

Clasa are următoarele metode:

* **public Monom(int grad, double coef):** constructor
* **public void setNegativ():** setează coeficientul monomului prin care e apelată să fie negativ
* **public Monom divide(Monom m):** împarte monomul prin care e apelată metoda cu un alt monom, primit ca parametru. Returnează un monom rezultat.
* **public int compareTo(Monom m):** compară gradul a două monoame și returnează diferența dintre gradurile acestora (return m.getGrad() - this.getGrad(); )
* **gettere, settere**

Clasa Monom implementează interfața Comparable. Acest lucru ne va fi util atunci când Polinomul va trebui sortat descrescător după grad. Este obligatorie să apară metodele interfeței și în clasa care o implementează. În cazul nostru, metoda **compareTo** e această metodă.

* Clasa **Polinom**:

Această clasă implementează obiectul Polinom, obiectul esențial proiectului nostru, deoarece pe aceste obiecte se vor efectua operațiile. Un polinom conține mai multe monoame. Clasa are un singur câmp:

* **List <Monom> polinom**: o listă (mai exact ArrayList) care conține monoamele.

Clasa are următoarele metode:

* **public void adaugaMonom(Monom m):** adaugă un monom primit ca parametru la lista de monoame a polinomului
* **public void stergeMonom(Monom m):** șterge un monom primit ca parametru din lista de monoame a polinomului
* **public void afisarePolinom():** afișează un polinom în consolă
* **public String convertToString():** convertește un polinom într-un String. Parcurge monoamele polinomului și le adaugă pe fiecare unui String inițial gol (String s=""), concatenându-le și separându-le cu un "+". Această metodă va fi folosită când va trebui să afișăm rezultatul polinom în căsuța de rezultat.
* **public void calcule():** parcurge lista de monoame a unui polinom, și, dacă găsește două monoame de același grad, le adună. Astfel, fiecare polinom va avea un singur termen la un anumit grad.
* **public void splitPolynom (String polinom):** pe această metodă se bazează funcționarea calculatorului. Utilizatorul introduce polinomul/polinoamele în căsuța acestora în format de String. Acest String este parametrul metodei. Această metodă practic transformă stringul într-un obiect Polinom, ca să se poată efectua operațiile pe el. Parcurge Stringul și se despart componentele prima dată dupa semnul "+", apoi după semnul "X^".
* **public Monom getMonomMax():** returnează monomul cu cel mai mare grad dintr-un polinom.
* **public int getGradMax():** returnează cel mai mare grad existent dintr-un polinom.
* **public void sortareDupaGrad():** sortează monoamele polinomului descrescător după grad.
* Clasa **Operation**:

Această clasă conține operațiile matematice ale polinoamelor. Astfel, clasa Operation nu are nici un câmp.

Clasa conține următoarele metode:

* **public Polinom adunarePolinoame(Polinom p1, Polinom p2):** efectuează adunearea a două polinoame. În metodă se declară două polinoame rezultat. Prima dată se parcurge cu un foreach primul polinom și se adaugă monoamele acestuia în polinomul rezultat1. Apoi se parcurge al doilea polinom tot cu foreach și se adaugă la fel și monoamele acestuia în rezultat1. În final, se apelează pe polinomul rezultat metoda de **calcule(),** care adună monoamele de același grad. În cazul în care prin adunare, unele elemente se elimină (ex. 2X^1 + (-2X^1)), coeficientul va fi zero. Ca să prevenim prezența acestor termeni în polinomul rezultat, parcurgem acesta cu un foreach și doar monoamele cu coeficientul diferit de zero se vor adăuga în polinomul rezultat2, care va fi returnat.
* **public Polinom scaderePolinoame(Polinom p1, Polinom p2):** efectuează scăderea a două polinoame (din p1 scade p2). În metodă se declară două polinoame rezultat (rezultat1, rezultat2). În primul polinom rezultat se adaugă monoamele lui p1, cu "+", apoi se adaugă și monoamele lui p2, dar acestea cu semnul "-" (se apelează pe ei metoda setNegativ() din clasa Monom). La final, se efectuează calculele pe monoamele de același grad în rezultat1, iar în rezultat2 se adaugă doar elementele cu coeficientul diferit de zero. Se returnează rezultat2.
* **public Polinom inmultire(Polinom p1, Polinom p2):** efectuează înmulțirea a două polinoame. Cu 2 foreach se înmulțesc toate elementele lui p1 cu toate elementele lui p2. Pe polinomul rezultat se apelează metoda de calcule, iar apoi acesta va fi returnat.
* **public Polinom derivare(Polinom p):** derivează polinomul primit ca parametru.
* **public Polinom integrare(Polinom p):** integrează polinomul primit ca parametru.
* **public Polinom inmultireCuMonom(Polinom p, Monom monom):** înumlțește polinomul p cu un monomul monom, primiți ca parametru. Fiecare grad al monoamelor polinomului p se adună cu gradul monomului și fiecare coeficient al monoamelor se înmulțesc cu coeficientul monomului.
* **public List<Polinom> impartire (Polinom p1, Polinom p2):** împarte polinomul p1 cu polinomul p2. Metoda returnează o listă de polinoame, care va conține mai exact 2 elemente: câtul și restul împărțirii.
* Clasa **View:**

În această clasă este implementată interfața utilizatorului. Clasa extinde JFrame. Are următoarele câmpuri:

* **JFrame frame**: fereastra calculatorului polinomial.
* **JTextField tf1:** căsuța în care utilizatorul introduce primul polinom.
* **JTextField tf2:** căsuța în care utilizatorul introduce al doilea polinom.
* **JTextField tf3:** căsuța polinomului rezultat (aici va fi afișat rezultatul operațiilor)
* **JTextField tf4:** căsuța polinomului rest (aici va fi afișat doar restul operației de împărțire, în restul cazurilor nu se afișează nimic)
* **JButton adunare:** butonul de adunare, "+"
* **JButton scadere:** butonul de scădere, "-"
* **JButton inmultire:** butonul de înmulțire, "×"
* **JButton impartire:** butonul de împărțire, "/"
* **JButton derivare:** butonul de derivare, " ’ "
* **JButton integrare:** butonul de integrare, " ∫ "
* **JButton clear:** butonul de clear, care șterge conținutul tuturor căsuțelor, "C"
* **Controller controller:** un obiect de tip controller, care face legătura între apăsarea butoanelor din View și acțiunile care trebuie să urmeze (implementate în Controller).

Clasa are următoarele metode:

* **Public View():** constructor, aici sunt implementate elementele de design ale calculatorului. Un astfel de element de design ar fi setarea mărimii ferestrei calculatorului ( frame.setSize(400, 250); ). Panelul principal al ferestrei este adăugat din 5 panele (panel1, panel2, panel3, panel4, panel5). Fiecare panel are propriile sale componente. Panel1 conține căsuța primului polinom, precedat de JLabelul label0 ("*Polynom 1:* "). Panel2 conține căsuța celui de al doilea polinom, precedat de JLabelul label4 ("*Polynom 2:* "). Panel1 conține căsuța primului polinom, precedat de JLabelul label0 ("*Polinom 1:* "). Panel3 conține butoanele de operație și butonul de clear. Panel4 conține căsuța polinomului rezultat, precedat de JLabelul label5 ("*Result:* "). Panel5 conține căsuța restului de la împărțire, precedat de JLabelul label6 ("*Rest:* "). Toate aceste paneluri sunt adăugate panelului principal p. (ex. p.add(panel1); ). Un alt aspect foarte important al calculatorului este adăugarea actionListener-urilor la butoanele de pe interfață, făcând legătura cu controllerul declarat ca și field (ex. *adunare.addActionListener(controller);* ).
* **public String getFirstPolynom():** returnează Stringul introdus în textfieldul tf1.
* **public String getSecondPolynom():** returnează Stringul introdus în textfieldul tf2.
* **public String setResult(String result):** afișează în căsuța de rezultat (tf3) Stringul primit ca și parametru.
* **public void setRest(String result):** afișează în căsuța de rest (tf4) Stringul primit ca parametru.
* **public void clearTextField():** șterge conținutul fiecărei căsuțe. Face acest lucru prin afișarea unui string gol("") în acestea.
* Clasa **Controller:**

În această clasă sunt implementate acțiunile care trebuie să urmeze după apăsarea unui buton. Ea implementează interfața ActionListener. Are următoarele câmpuri:

* **View view**
* **Operation operation**

Clasa are următoarele metode:

* **public Controller(View view):** constructor
* **public void actionPerformed(ActionEvent e):** metodă specifică interfeței ActionListener, ce trebuie implementată în mod obligatoriu. În această metodă se verifică cu câte un if, care buton s-a apăsat, și în funcție de butonul apăsat, se execută acțiunile necesare. Ca de exemplu, dacă s-a apăsat butonul adunare (*if(source == view.adunare)*) , atunci controllerul va lua Stringurile introduse în căsuțele tf1 și tf2, le va converti în polinoame aplicând metoda *splitPolinom(String s),* iar apoi va efectua operația de adunare pe ele. Pe urmă, va afișa rezultatul operației în căsuța de rezultat, folosind metoda setResult. Controllerul așa va proceda și la restul butoanelor.
* Clasa **Tests:**

În această clasă se testează operațiile matematice asupra polinoamelor folosind frameworkul JUnit. Metodele din această clasă sunt teste, și sunt precedate de anotația @Test. Clasa are următorul câmp:

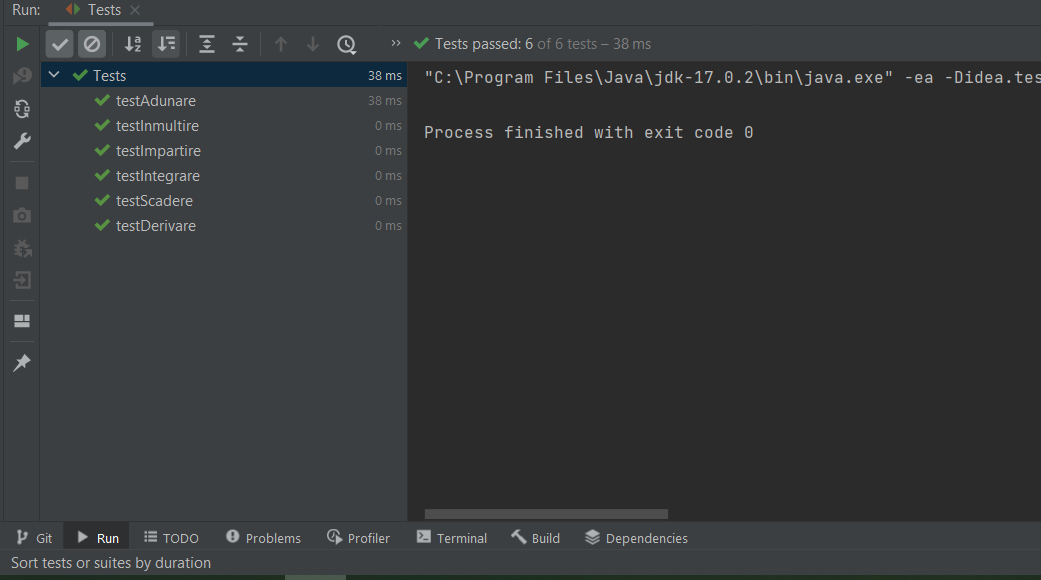
* **Operation operation**

Are următoarele metode:

* **public void testAdunare():** testează adunarea a două polinoame. Convertește polinomul rezultat obținut într-un string și îl compară cu rezultatul corect (care este un String) cu metoda assertEquals().
* **public void testScadere():** testează scăderea a două polinoame. Convertește polinomul rezultat obținut într-un string și îl compară cu rezultatul corect (care este un String) cu metoda assertEquals().
* **public void testDerivare():** testează derivarea unui polinom. Convertește polinomul rezultat obținut într-un string și îl compară cu rezultatul corect (care este un String) cu metoda assertEquals().
* **public void testIntegrare():** testează integrarea unui polinom. Convertește polinomul rezultat obținut într-un string și îl compară cu rezultatul corect (care este un String) cu metoda assertEquals().
* **public void testInmultire():** testează înmulțirea a două polinoame. Convertește polinomul rezultat obținut într-un string și îl compară cu rezultatul corect (care este un String) cu metoda assertEquals().
* **public void testImpartire():** testează împărțirea a două polinoame. Compară cu un for fiecare element al polinomului rezultat obținut cu polinomul rezultat corect. Face acest lucru prin compararea coeficienților și gradurilor fiecărui polinom cu metoda assertEquals().

# 5. Rezultate

Testarea funcționalităților proiectului a fost realizată în clasa ***Tests*** cu frameworkul de testare JUnit. Trebuia importată clasa JUnit Assert (*import static org.junit.Assert.\*; import org.junit.\*;*), care era responsabilă pentru efectuarea testelor.



Pentru fiecare operație matematică s-a efectuat un test: *testAdunare()*, *testScadere(), testInmultire(), testImpartire(), testDerivare(), testIntegrare()* . Un test reprezintă o metodă precedată de anotația **@Test**, în care se verifică rezultatul obținut de la operația respectivă cu rezultatul corect introdus manual de către programator. Verificarea se face cu metoda **assertEquals()**, metodă specifică clasei Assert, care verifică dacă două obiecte sunt egale sau nu.



***Exemplu: testarea operatiei de adunare (metoda testAdunare( ))***

În fiecare metodă de testare am introdus manual, în format de string, polinoamele pe care se efectuează operațiile și rezultatul corect. Am efectuat operația respectivă pe polinoame și am salvat rezultatul. Am convertit rezultatul într-un String, după care l-am comparat cu rezultatul corect (care este deja un String) cu metoda assertEquals().

Restul testărilor:











# 6. Concluzii

Ca orice proiect sau aplicație, și acesta a avut dificultățile lui, din care se pot trage concluzii și idei, care ne pot fi utile în implementarea altor proiecte în viitor. Un lucru important ar fi că, pentru realizarea unei obiective, cerințe, trebuie mai întâi să găsim soluții sub-obiectivelor. Un obiectiv principal e alcătuit din mai multe sub-obiective. O altă concluzie ar fi că rezolvarea unei probleme impune programatorului să găsească structurile de date cele mai potrivite pentru caz. În cazul nostru, folosirea ArrayListului a fost mult mai optim decât folosirea unui vector de obiecte. Tot așa, folosirea foreach-ului a fost mai ușoară și straight-forward decât folosirea unui for obișnuit. În final, o altă chestie importantă ar fi importanța lizibilității codului, folosind metode nu mai lungi de 30 de rânduri și clase nu mai lungi de 300 de rânduri.

O dezvoltare ulterioară a calculatorului ar putea fi afișarea unui mesaj utilizatorului în cazul în care acesta nu a introdus polinomul corect.

# 7. Bibliografie

<https://www.vogella.com/tutorials/JUnit/article.html>

<https://docs.oracle.com/javase/tutorial/uiswing/>

<https://dsrl.eu/courses/pt/materials/A1_Support_Presentation.pdf>