**DOCUMENTATIE**

**TEMA 1**

NUME STUDENT: Mîrza Andrei Tudor

GRUPA: 30224

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 4](#_Toc95297886)

[3. Proiectare 5](#_Toc95297887)

[4. Implementare 6](#_Toc95297888)

[5. Rezultate 6](#_Toc95297889)

[6. Concluzii 7](#_Toc95297890)

[7. Bibliografie 8](#_Toc95297891)

**1.Obiectivul temei.**

Obiectivul principal al temei este de a realiza un calculator care executa cateva operatii pentru polinoame si de a exersa mai in detaliu metodele de programare in Java.

Obiectivele principale reprezinta pasii care trebuie urmati pentru indeplinirea obictivului principal si anume:

* Intelegerea cerintelor propuse
* Scrierea in cod a ideilor de rezolvare
* Testarea calculatorului creat

**2.Analiza problemei, modelare, scenarii, cazuri si utilizare**

Analiza problemei : -Operatiile pe polinoame sunt operatii matematice nu foarte greu de inteles, dar care necesita un minim de cunostiinte in acest domeniu.

Modelarea acesteia se refera la modul in care interfata a fost creeata . Cu cat aceasta este mai simpla cu atat orice utilizator va putea folosi aplicatia mult mai usor.

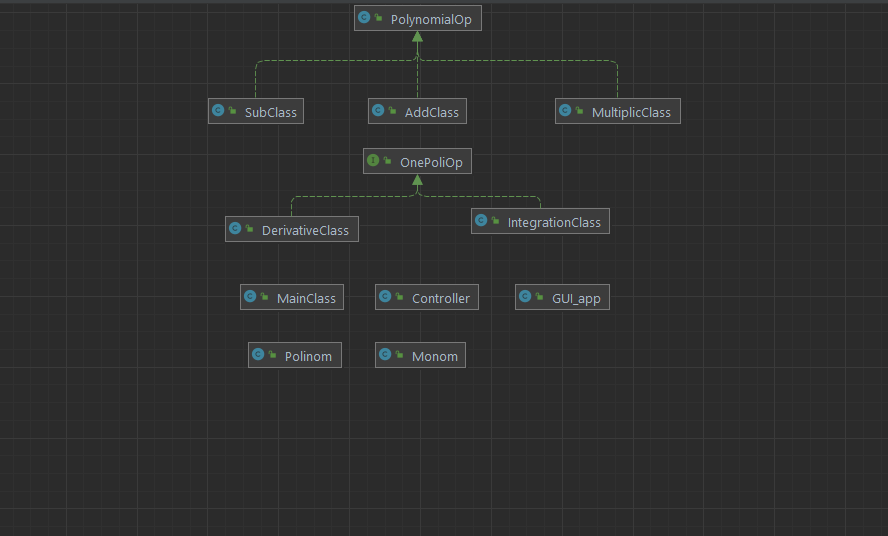
Cazurile de utilizare sunt:

-avem doua campuri in care utilizatorul trebuie sa introduca polinoamele, in cazul in care se intrduce un polinom gresit va aparea un mesaj de eroare

-apoi exista 6 butoane prin care se vor executa operatiile de adunare, scadere, inmultire, impartire, derivare si integrare

-rezultatul se va afisa in chenarul de mai jos tot sub forma unui polinom

**3.Proiectare**



Aici avem diagrama UML de clase si pachete. Dupa cum putem observa am creat o interfata pentru operatille care se executa cu ambele polinoame si anume adunarea, scaderea si inmultirea. Apoi am creat inca o interfata pentru operatiile pe un singur polinom si anume derivarea si integrarea, ambele se vor aplica pe polinomul numarul 1.

Arhitectura folosita este Model-View-Controller deoarece separa reprezentarea informatiilor din interactiunea cu utilizatorul in sine si anume:

* Utilizatorul interactioneaza cu interfata de utilizator ( de exemplu apasa un buton)
* Controller-ul preia datele adaugate de catre utilizator
* Controller- ul creaza modelul, eventual il actualizeaza intr-un mod adecvat pentru actiunea utilizatorului si este trimis inapoi la view
* View – ul utilizeaza modelul pentru a genera o interfata pentru utilizator adecvata cerintelor, in cazul nostru avand in interfata doua chenare pentru inserarea celor doua polinoame , 6 butoane, unul pentru fiecare operatie si inca un chenar unde se va afisa rezultatul.

Pentru stocarea monoamelor am folosit ArrayList- uri , iar polinoamele le-am creat in HashMap - uri pentru a le putea parcurge mai usor si pentru a efectua operatiile pe ele mult mai simpu si rapid. Monoamele le – am sortat dupa grad in ordine descrescatoare pentru a aplica mai eficient operatiile.

**4. Implementare**

Toate clasele pe care le – am creat sunt structurate in package – uri pentru o navigare mai usoara intre ele si pentru o intelegere mai usoara a functionalitatii fiecareia dintre ele.

In pachetul GUI avem doua clase si anume “Controller” si “GUI\_app”.

In clasa “Controller” sunt atribuite operatiile pentru fiecare buton, avand control asupra fiecaruia. Aici avem 5 metode :

* Cea pentru adunare “AddPackage”
* Cea pentru scadere “SubbPackage”
* Cea pentru derivare “DerivativePackage”
* Cea pentru integrare “IntegrativePackage”
* Cea pentru inmultire “MultiplicPackage”

Fiecare dintre metodele mentionate mai are la baza aceeasi metoda de functionare, adica adaugarea unei anumite actiuni asupra unui buton.

In clasa “GUI\_app” este creata interfata grafica care contine 3 TextField – uri, in primele doua se introduc cele doua polinoame iar in ultimul se afiseaza rezultatul. Pe langa aceste TextField – uri avem si cele sase butoane, unul pentru fiecare operatie.

Urmatorul pachet este “Model” in care avem clasele “Monom” si “Polinom”.

In clasa “Monom” sunt prezentate puterea si coeficientul fiecaruia. In metoda “equals” se verifica daca avem doua monoame cu aceeasi putere.

In clasa “Polinom” avem un ArrayList de monoame sortate descrescator . In aceasta clasa o metoda importanta este “transformP” prin intermediul care se transforma string ul introdus intr-un monom pentru a putea sti care este putera si coeficientul.O alta metoda importanta in aceasta clasa este “toString” prin intermediul careia afisam rezultatul obtinut in urma operatiilor effectuate. Aceasta metoda va afisa in cazul in care rezultatul este diferit de zero un nou polinom.

Urmatorul pachet este pachetul de operatii, avand numele “Operatii” care contine clasele: “AddClass”, “DerivativeClass”, “IntegrationClass”, “MultiplicClass”, “SubClass”, “MainClass” si cele doua interfete “OnePoliOp” si “PolynomialOp”.

1.AddClass – aceasta clasa implementeaza interfata “PolynomialOp” deoarece operatia de adunare se realizeaza intre doua cele doua polinoame. In aceasta clasa avem doua metode , in prima metoda avem ca parametrii cele doua polinoame pe care le introducem intr – un HashMap . Dupa avem apelul celei de – a doua metode prin care introducem monoamele in HashMap adaugandu – le mai apoi in polinom. Cea de-a doua metoda “putInto”, primeste ca si parametrii un HashMap si un polinom. Aici se parcurge HashMap – ul si se verifica daca avem monoame de acelasi grad, daca am gasit doua monoame de acelasi grad le adunam, altfel adaugam monomul si se trece la urmatoarele verificari pana se ajunge la final.

2.DerivativeClass – aceasta clasa implementeaza interfata “OnePoliOp” deoarece operatia de derivare se face doar pe un polinom, in cazul nostrum aceasta se va face pe primul polinom introdus de catre utilizator. In aceasta clasa avem doua metode . Prima metode se numeste “forAll1” care primeste ca si parametru polinomul ce trebuie derivate. Mai apoi avem apelul celei de – a doua metode prin care introducem monoamele intr – un HashMap adaugandu – le maim ai apoi in polinom. In cea de – a doua metoda avem ca si parametrii un HashMap si un polinom, aici se parcurge lista de monoame si derivam monomul doar daca puterea acestuia este diferita de zero.

3.IntegrationClass – aceasta clasa implementeaza interfata “OnePoliOp” deoarece operatia de integrare se face doar pe un polinom, in cazul nostrum aceasta operatie se va face pe primul polinom introdus de catre utilizator. In aceasta clasa avem doua metode. Prima metoda se numeste “forAll1” care primeste ca si parametru polinomul ce trebuie integrat. Dupa avem apelul celei de – a doua metode prin care introducem monoamele intr – un HashMap adaugandu – le mai apoi in polinom, In ce a de – a doua metode avem ca si parametrii un HashMap si un polinom , aici spre deosebire de adunare si scadere nu mai este necesar sa facem atatea verificari ci doar sa verificam daca puterea este diferita de zero, in cazul in care se intampla acest lucru integram monomul si trecem la urmatorul, iar in cazul in care puterea este zero nu se aplica integrarea .

4.MultiplicClass – aceasta clasa implementeaza interfata “PolynomialOp” deoarece operatie de inmultire se realizeaza intre doua polinoame.Aici avem doar o metoda spre deosebire de adunare si scadere unde aveam doua cu toate ca aceasta operatie se aplica pe doua polinoame. Aceasta metode se numeste “forAll” care primeste ca si parametrii doua polinoame . Pentru fiecare in parte se parcurge lista de monoame , iar pentru prima data se aduna puterile si se inmultesc coeficientii. Noul rezultat va fi returnat.

5.SubClass - aceasta clasa implementeaza interfata “PolynomialOp” deoarece operatia de scadere se realizeaza intre doua cele doua polinoame. In aceasta clasa avem doua metode , in prima metoda avem ca parametrii cele doua polinoame pe care le introducem intr – un HashMap . Dupa avem apelul celei de – a doua metode prin care introducem monoamele in HashMap adaugandu – le mai apoi in polinom. Cea de-a doua metoda “putInto”, primeste ca si parametrii un HashMap si un polinom. Aici se parcurge HashMap – ul si se verifica daca avem monoame de acelasi grad, daca am gasit doua monoame de acelasi grad le scadem, altfel adaugam monomul si se trece la urmatoarele verificari pana se ajunge la final.

Si in cele din urma in clasa “MainClass” avem apelurile fiecarei clase mentionate mai sus.

**5.Rezultate**

Pentru a testa operatiile am folosit JUnit .

Junit este un cadru de testare unitare pentru limbajul de programare Java, JUnit a fost important în dezvoltarea dezvoltării bazate pe teste și face parte dintr-o familie de cadre de testare unitară care este cunoscută în mod colectiv sub numele de xUnit , care a apărut cu SUnit .

JUnit este conectat ca JAR în timpul compilării. Cea mai recentă versiune a cadrului, JUnit 5, se află sub pachetul org.junit.jupiter. Versiunile anterioare JUnit 4 și JUnit 3 erau sub pachete org.junitși junit.framework, respectiv. Testarea unitară s-a impus în ultima perioadă în dezvoltarea proiectelor scrise în limbajul Java şi numai, pe măsura apariţiei unor utilitare gratuite de testare a claselor, care au contribuit la creşterea vitezei de programare şi la micşorarea drastică a numărului de bug-uri.se îmbunătăţeşte viteza de scriere a codului, concomitent cu creşterea calităţii acestuia, deoarece prin scrierea testelor unitare se micşorează timpul de depanare, pemiţând refactorizarea mai uşoară, cu depistarea imediată a eventualelor erori inserate în codul modificat;

Clasele de test sunt uşor de scris şi modificat pe măsură ce codul sursă se măreşte, putând fi compilate împreună cu codul sursă al proiectului. Compilatorul testează sintaxa codului sursă, în timp ce clasele de test validează integritatea codului

Clasele de test JUnit pot fi rulate automat (în suită), rezultatele fiind vizibile imediat. Se pot crea ierarhii de suite de test, care pot fi testate împreună sau separat, în funcţie de cerinţete proiectului. Ca efect secundar al utilizării pe scară largă, versiunile anterioare ale JUnit rămân populare, JUnit 4 având peste 100.000 de utilizări de către alte componente software din depozitul central Maven .

Avantaje pentru utilizarea testarii unitare:

- Scopul testării unitare este de a izola fiecare parte a programului și de a arăta că părțile individuale sunt corecte.  Un test unitar oferă un contract strict, scris, pe care fragmentul de cod trebuie să îl îndeplinească. Drept urmare, oferă mai multe beneficii.

- Testarea unitară descoperă probleme la începutul ciclului de dezvoltare . Aceasta include atât erori în implementarea programatorului, cât și defecte sau părți lipsă ale specificației pentru unitate. Procesul de scriere a unui set amănunțit de teste îl obligă pe autor să se gândească la intrări, ieșiri și condiții de eroare și, astfel, să definească mai clar comportamentul dorit al unității. Costul găsirii unei erori înainte de a începe codarea sau atunci când codul este scris pentru prima dată este considerabil mai mic decât costul detectării, identificării și corectării erorii mai târziu. Erorile din codul lansat pot cauza, de asemenea, probleme costisitoare pentru utilizatorii finali ai software-ului

- Testarea unitară permite programatorului să refactorizeze codul sau să actualizeze bibliotecile de sistem la o dată ulterioară și să se asigure că modulul încă funcționează corect (de exemplu, în testarea regresiei ). Procedura este de a scrie cazuri de testare pentru toate funcțiile și metodele, astfel încât ori de câte ori o modificare provoacă o defecțiune, aceasta să poată fi identificată rapid. Testele unitare detectează modificări care pot rupe un contract de proiectare .

- Atunci când software-ul este dezvoltat folosind o abordare bazată pe teste, combinația de scriere a testului unitar pentru a specifica interfața plus activitățile de refactorizare efectuate după ce testul a trecut, poate înlocui proiectarea formală. Fiecare test unitar poate fi văzut ca un element de proiectare care specifică clase, metode și comportament observabil.

Dezavantaje:

- Testarea nu va prinde fiecare eroare din program, deoarece nu poate evalua fiecare cale de execuție în niciunul dintre programele cele mai banale. Această problemă este un supraset al problemei opririi , care este indecidabilă . Același lucru este valabil și pentru testarea unitară. În plus, testarea unitară prin definiție testează doar funcționalitatea unităților în sine. Prin urmare, nu va detecta erori de integrare sau erori mai ample la nivel de sistem (cum ar fi funcțiile efectuate în mai multe unități sau zonele de testare nefuncționale, cum ar fi performanța ).

**6.Concluzii**

In concluzie din aceasta tema am invatat utilizarea arhitecturii MVC si multe alte tehnici de programare in limbajul de programare Java.

**7.Bibliografie**

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

2. <https://regex101.com/>

3. https://netbeans.apache.org/kb/docs/java/gui-functionality.html