**DOCUMENTATIE**

**Tema Numărul 1**

Nume: Ilovan Mara Gabriela

Grupa: 30222

**CUPRINS**

1. Obiectivul temei..........................................................................................3
2. Analiza problemei, modelare, scenarii, cazuri de utilizare.........................4
3. Proiectare.....................................................................................................6
4. Implementare...............................................................................................7
5. Rezultate......................................................................................................9
6. Concluzii...................................................................................................10
7. Bibliografie...............................................................................................11

**Obiectivul temei**

Obiectivul principal al temei este implementarea unui calculator de polinoame, care poate fi folosit prin intermediul unei interfețe grafice intuitive care sa permită utilizatorilor sa insereze polinoame, sa selecteze operația dorita si sa vizualizeze rezultatul calculului matematic. Astfel, se pot: aduna, scădea, înmulții sau deriva doua polinoame si deriva sau integra cate un polinom. Aceste operații ar trebui sa fie ușor accesibile prin intermediul interfeței grafice si sa ofere rezultate corecte. In plus, calculatorul ar trebui sa îmbunătățească cat mai mult experiența utilizatorului. Se pot include metode pentru validarea datelor pentru a ajuta utilizatorul in cazul in care introduce polinoamele greșit.

Alte obiective secundare au constat in: analizarea unei probleme si identificarea cerințelor, proiectarea calculatorului polinomial- realizarea unui plan in ceea ce privește funcționarea unui calculator de polinoame, implementarea calculatorului- dezvoltarea codului, testarea calculatorului de polinoame, verificarea corectitudinii operațiilor realizate. Testarea e realizata prin compararea rezultatelor cu rezultatele așteptate.

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

Cerințele funcționale sunt acelea care descriu funcționalitățile pe care programul trebuie sa îndeplinească, in timp ce cerințele non funcționale sunt cerințele care descriu programul.

Cerința problemei este: crearea unei interfețe grafice, care sa permită utilizatorului sa introducă doua polinoame pentru: adunare, scădere, înmulțire, împărțire si unul pentru derivare si integrare. Operațiile pot fi selectate din interfața, algoritmii acestora fiind esențiali pentru funcționare. Rezultatele operațiilor vor fi disponibile in caseta text corespunzătoare rezultatului, după apăsarea butonului potrivit. Calculatorul trebuie sa ofere funcții de calcul pentru operațiile menționate mai sus.

Alte aspecte care descriu calculatorul de polinoame sunt: performanța – acesta ar trebui sa execute comenzile rapid, fiind prietenos cu utilizatorul, fiabilitatea – sa nu apară erori in timpul calculelor matematice, programul ar putea avea diferite modificări ulterior prin adăugare de operații noi sau îmbunătățirea celor existente. Calculatorul trebuie sa fie ușor de folosit de către utilizator.

In funcție de operație calculatorul are trei întări sau doua. In cazul derivării si integrării au doar un polinom pe care se realizează calculul, iar in cadrul celorlalte operații e nevoie de doua polinoame. Ultima intrare consta in operația dorita. Calculatorul are in fiecare caz o ieșire, exceptând împărțirea, unde rezultatul e împărțit in cat si rest.

Funcționalitatea aplicației e descrisa de diferite cazuri de utilizare a acesteia. In ceea ce privește calculatorul de polinoame funcționalitatea e redata de operațiile matematice implementate astfel încât rezultatul final sa fie unul dorit.

O imagine care conține diagramă

Descriere generată automat

Cazurile de utilizare sunt reprezintă chiar operațiile. Astfel, utilizatorul poate alege ce urmează sa facă. Scenariul pe care trebuie sa îl urmeze consta in:

* Inserarea inputurilor in interfața grafica
* User-ul apasă butonul corespunzător operației dorite: scădere, adunare, înmulțire, împărțire, derivare sau integrare
* Calculatorul de polinoame calculează rezultatul si îl transmite in interfața, pentru ca utilizatorul sa poată vedea.
* Daca polinoamele sunt introduse greșit se va afișa mesajul: „Invalid input. Please try again!”
* Scenariul se poate întoarce la primul pas

**Proiectare**

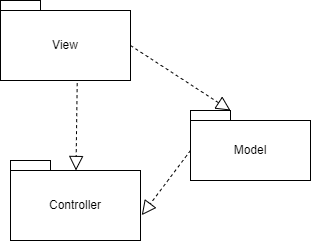
Programarea orientata pe obiecte este o paradigma de programare ce are la baza conceptul obiectelor- unități de programare care conțin funcționalități asociate. Astfel, este permisa manipularea de obiecte pentru crearea mai multor sarcini in cadrul unui proiect. Acest tip de programare folosește conceptul de încapsulare al datelor, care permite ascunderea detaliilor unui obiect, fapt ce duce la protejarea datelor. Pentru a construi un program eficient, ușor de întreținut, cu o structura clasica si o organizare logica a datelor la prima tema vom folosi principii OOP.

Astfel, pentru acest proiect am ales sa folosesc doua clase:

1. Clasa Monomial: Un monom este o parte a polinomului. Acesta este caracterizat de un coeficient – double (pentru integrare si împărțire) si un exponent întreg. Clasa Monomial este necesara pentru a ușura munca programatorului la dezvoltarea operațiilor.
2. Clasa Polynomial: Aceasta clasa conține un Map de monoame, deoarece aceasta au o căutare foarte rapida, cheia, va fi exponentul întrucât e unic.

Pentru o mai buna lizibilitate a codului proiectul va fi împărțit in mai multe pachete:

* View, in care vom avea clasele necesare pentru implementarea interfeței grafice
* Model, care conține clasele ce modelează datele aplicației
* Controller, care controlează interfața (face legătura cu operațiile)



In pachetul Model vom avea calsele Polynomial si Monomial. De asemenea, clasa Arithmetic este si ea in acest pachet. Aici se vor implementa operațiile. Astfel, avem nevoie de metode pentru: adunare, scădere, înmulțire, împărțire, derivare si integrare. Ca metode intermediare vom folosi metode ce fac operații pe monoame

In pachetul View vom avea clasa PolynomialCalculatorView care va implementa modelul sub forma dorita. Vom avea nevoie de doua TextField-uri pentru introducerea de polinoame, si unul pentru rezultat. Totodată sunt necesare 6 butoane, care vor avea numele fiecărei operații.

In pachetul Controller vom avea o clasa responsabil pentru răspunsul aplicației la intrările introduse de utilizator. In funcție de ce utilizatorul decide vom vedea răspunsul in interfața grafica.

**Implementare:**

Clasa Monomial: Aceasta clasa are ca atribute care un coeficient si un exponent. Coeficientul am ales sa fie double pentru împărțire si integrare, unde sunt posibile rezultate cu coeficient rațional, nu doar întreg. Exponentul a fost ales întreg. Deoarece am folosit încapsularea datelor implementarea de settere si gettere este necesara, totodată avem nevoie de de un constructor.

Clasa Polynomal: Aceasta clasa are ca atribut un TreeMap de monoame, cheia fiind exponentul, iar valoarea tot monomul. Am ales TreeMap întrucât datele se stochează in ordine, fără metode auxiliare pentru a realiza aceasta ordonare a monoamelor. Pentru adăugarea in map avem o metoda care mai întâi verifica daca monomul cu gradul dat a mai fost introdus in polinom, daca a fost introdus îl aduna la elementul cu gradul respectiv, altfel îl adaugă la map. Pentru schimbarea unui string in polinom avem o metoda verifyRegex, care verifica daca stringul a fost introdus corect, returnând false in caz contrar, iar daca stringul a fost introdus corect se va realiza transformarea. La realizarea împărțirii stringului am avut nevoie de cunoștințe in privirea regex (regular expression). Metoda toString pentru transformarea polinomului in string a fost implementata tot in clasa Polynomial.

Clasa Arithmetic: implementează toate operațiile specifice polinoamelor, astfel avem cate o metoda destinata pentru fiecare dintre operații. Totodată, metodele care realizează operațiile pe monoame se afla tot in aceasta clasa.

* **add(Polynomial p1, Polynomial p2)** – face adunarea a doua polinoame

Pentru aceasta metoda a font nevoie sa ne gândim la toate cazurile posibile: polinoamele conțin doar monoame care au corespondent (in cea ce privește gradul) in al doilea polinom, primul polinom are monoame ce nu sunt in celălalt, al doilea are monoame ce nu apar in primul sau polinoamele nu au niciun exponent comun. Astfel, pe lângă însumarea coeficienților cu exponent comun a fost nevoie de căutare de monoame ce nu au corespondent si introducerea lor in polinom

* **sub(Polynomial p1, Polynomial p2)** – face sacadarea dintre cele doua polinoame

Algoritmul scăderii e asemănător cu cel pentru adunare. Diferit este faptul ca făcând diferența in loc de suma de monoame, coeficientul poate fi 0. In acest caz monomul nu va mai fi adăugat la polinomul rezultat.

* **multiplication(Polymonial p1, Polymonial p2)** – face înmulțirea polinoamelor

Pentru realizarea acestui algoritm a fost nevoie doar de o trecere prin polinoame, in care înmulțim monom cu monom.

* **derivative(Polynomial p) –** degrevează un polinom

La derivarea unui monom coeficientul se înmulțește cu gradul, iar gradul scade cu unu. Pentru derivarea unui polinom, datorita faptului ca polinomul este o suma de monoame, a fost nevoie doar de derivarea fiecărui monom din polinom, acestea fiind adăugate rezultatului.

* **Integrate(Polynomial p)** – integrează polinomul

Integrarea este procesul invers derivării. Astfel daca integram un monom coeficientul îl împărțim la exponentul incrementat, iar noul exponentul va fi cel incrementat. După care integrarea se realizează pe aceeași idee cu derivarea.

* **division(Polynomial p1, Polynomial p2)** – realizează împărțirea polinoamelor

Împărțirea reprezintă cel mai complex algoritm implementat. Primul pas este împărțirea primului monom din primul polinom cu primul monom din al doilea, rezultatul va fi adăugat unei variabile in care stocam rezultatul. Pasul doi reprezintă înmulțirea rezultatului cu împărțitorul. In pasul al treilea trebuie sa scădem din polinomul deîmpărțit inițial rezultatul înmulțirii anterioare. Acum împărțitorul va fi înlocuit de scădere. Acești pași sunt realizați pana când gradul primului polinom e mai mic sau egal decât gradul celui de-al doilea. La final vom avea rezultatul, iar rezultatul va fi reprezentat de primul polinom.

Clasa PlolynomialCalculatorView: reprezintă interfața grafica a aplicației. Interfața e formata din doua câmpuri de text care permit introducerea de către utilizator a doua polinoame. Exista șase butoane cu nume sugestive, astfel încât utilizatorul sa poată alege ușor calculul dorit. De asemenea, avem un câmp de text si pentru rezultatul operației. Fiecare buton are asociat un action listener.

Clasa Controller: Aceasta clasa leagă modelul de view si gestionează toate acțiunile venite din partea utilizatorilor. In aceasta clasa avem mai multe clase interne care implementează interfața ActionListener. Fiecare clasa corespunde unui buton si conține o metoda actionPerformed, in care luam datele din interfața, realizam operația corespunzătoare, si afișam rezultatul.

**Rezultate**

Pentru a ne asigura ca rezultatele operațiilor sun corecte a trebuit sa facem mai multe teste. JUnit este un framework de testare unitara pentru Java, utilizat pentru testarea automata a codului. Programatorii pot scrie cod care verifica daca codul are rezultatul așteptat. In cadrul metodelor făcute pentru testarea operațiilor vom folosii aserțiuni care verifica corectitudinea outputului.

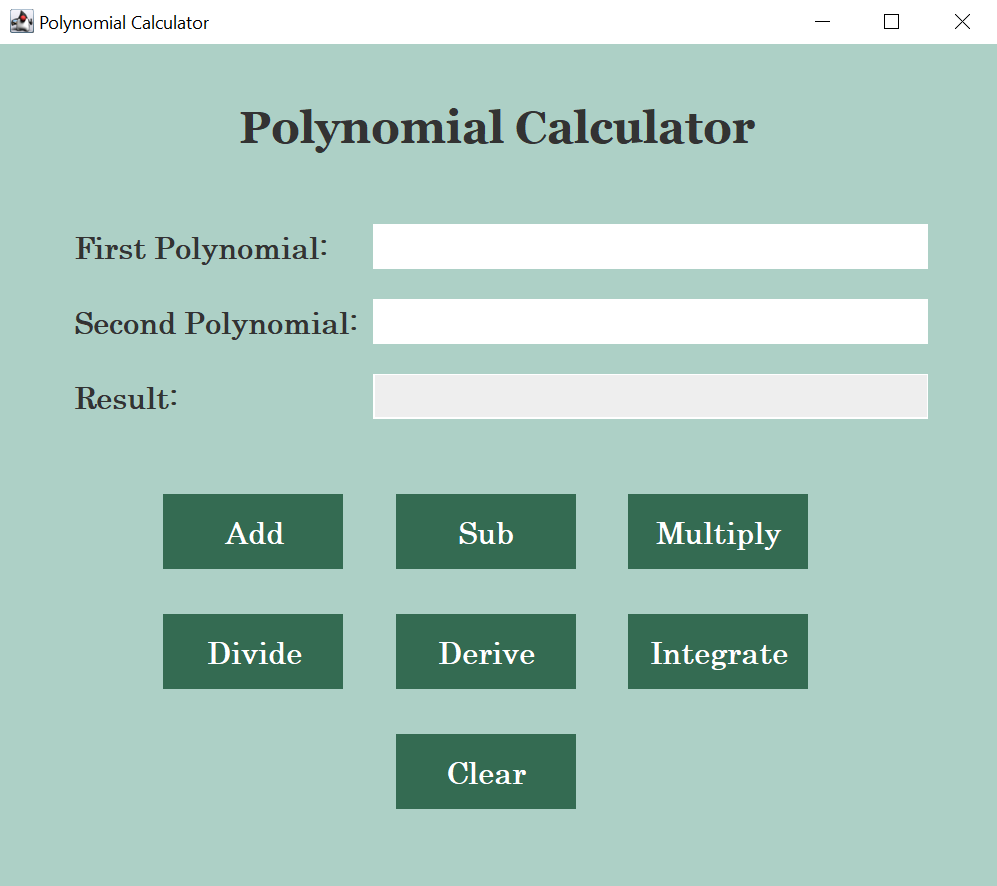
Pentru scrierea testelor am avut nevoie de o clasa separata fata de clasa ce conține codul sursa pe care dorim sa îl testam. Datorita proiectului de tip maven care oferă o execuție a testelor in mod automatizat. Prin acest fapt munca programatorului este ușurata, nefiind nevoie de execuția manuala.

Astfel, pentru adunare am testat daca polinoamele : x^3-2x^2+6x-5 si x^2-1 vor fi egale cu polinomul: x^3-3x^2+6x-4, iar rezultatul a fost afirmativ. La fel am făcut si pentru celelalte operații, rezultatul fiind corect pentru fiecare din metodele testate.

Totodată este necesara si testarea operațiilor in interfața pentru a ne asigura ca utilizatorul va beneficia de o experiența cat mai buna. Aici putem verifica corectitudinea metodei toString, pentru afișarea in caseta text, dar si transformarea stringului in polinom, in cazul in care este introdus greșit trebuie sa apară mesajul corespunzător. Tot aici vom verifica daca butonul clear funcționează corect, acesta pregătind interfața de introducerea de date noi.

Pentru a testa o aplicație cat mai bine trebuie sa ne gândim la toate cazurile posibile. De exemplu in cazul adunării a fost necesar sa testez aplicația pentru următoarele cazuri: ambele polinoame conțin doar monoame in comun, primul polinom poate conține monoame cu grad diferit, al doilea polinom poate conține monoame cu grad diferit, ambele polinoame pot avea monoame de grad diferit. După ce programul a reușit sa treacă toate testele, am știut ca algoritmul dezvoltat este corect.

La final aplicatia va arata asa, fiind pregatita de utilizare. In primele doua caeste se vor introduce polinoamele, dupa care apasam pe butonul dorit, iar rezultatul va aparea in a treia caseta.



**Concluzii**

Cu siguranța aceasta tema a fost una din care am învățat ceva. In primul rând, am învățat cum sa îmi structurez mai bine codul. Astfel, metoda utilizata a fost MVC (model, View, Controller), care împarte programul in: partea vizuala - interfața grafica, partea de logica (operațiile si modelul folosit – clasele Polynomial si Monomial) si partea care face legătura intre ele. Foșnind aceasta metodul codul devine mai lizibil.

De asemenea, am învățat cum se lucrează cu regex, cum găsesc un pattern, si cum împart mai apoi textul in fragmentele dorite. Deși a părut cea mai grea parte a proiectului, întrucât era ceva nou, problema a fost rezolvata relativ ușor.

Dar cel mai important lucru care l-am învățat lucrând la aceasta tema a fost faptul ca înainte de a scrie codul propriu zis trebuie sa ne gândim ce clase dorim sa folosim, la împărțirea pe pachete si la toate cazurile de utilizare si scenariile aplicației. Aceste lucruri trebuie făcute pentru a ne ușura scrierea de cod si sa evitam cat de mult posibil greșelile.

**Bibliografie**

1. Java - Regular Expressions: <https://www.tutorialspoint.com>
2. <https://stackoverflow.com/>
3. <https://www.cuemath.com/algebra/long-division-of-polynomials/>
4. Java Swing Tutorial: <https://www.javatpoint.com/>
5. Java AWT Tutorial: <https://www.javatpoint.com/>
6. <https://www.programiz.com/java-programming/treemap>