DOCUMENTAȚIE

TEMA 1

NUME STUDENT: Ghiorghe Sorana
GRUPA: 30225

CUPRINS

1. Obiectivul temei

Obiectiv principal

Obiectivul principal al temei este proiectarea și implementarea unui calculator polinomial alături de o interfață grafică prin care utilizatorul poate introduce două polinoame, selecta operația matematică de efectuat între ele și vizualiza rezultatul.

Obiectivele secundare

1.	Determinarea claselor necesare pentru implementarea calculatorului (capitolul 4) • Clasa Polinom cu structura de date HashMap pentru stocarea monoamelor și gestionarea fiecărui polinom. • Clasa de Operații care conține metode pentru realizarea operațiilor pe structura de date menționată.
2.	Crearea unei interfețe grafice (cap 4) • Două zone de input pentru polinoame, selecție a operației și zonă pentru rezultat.
3.	Gestiunea interacțiunii dintre interfață și polinoame • Utilizarea unui regex pentru traducerea inputului în structura de HashMap și verificarea corectitudinii procesului.
4.	Testare și depanare (cap 5) • Teste unitare utilizând JUnit și depistarea punctelor slabe neobservate în timpul implementării.
5.	Documentațe • Elaborarea unei documentații care să cuprindă detalii despre implementarea aplicației pentru o înțelegere corespunzătoare asupra calculatorului și modul în care funcționează..

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Problema abordată constă în dificultatea efectuării operațiilor matematice între polinoame manual. În acest context, aplicația propune dezvoltarea unui calculator care să faciliteze și să automatizeze procesul de efectuare a acestor operații, având ca rezultat o experiență mai simplă și mai eficientă pentru utilizatori.

Cerințe Funcționale

- Utilizatorul poate introduce polinoame de la tastatură, care respectă formatul specificat.
- Calculatorul trebuie să calculeze doar polinoame care respectă formatul cerut.
- Utilizatorul poate selecta operația matematică dorită, acestea fiind adunarea, scăderea, înmulțirea, împărțirea, derivarea și integrarea.

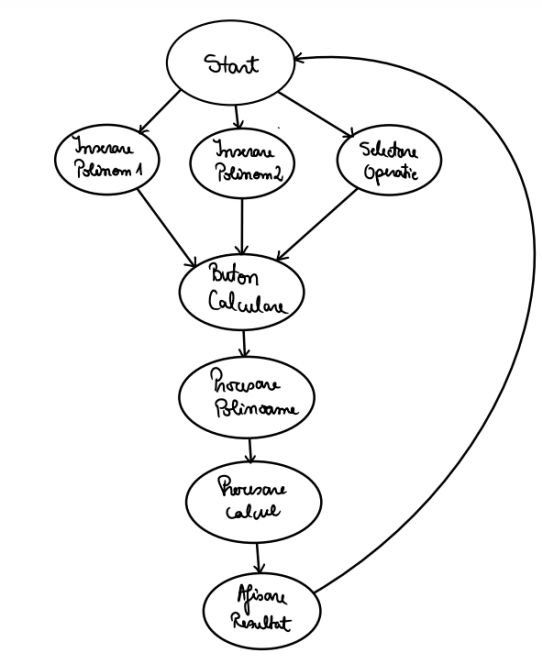
- Calculatorul trebuie să realizeze operația specificată între cele două polinoame dacă operația selectată include două polinoame ca input.
- Calculatorul trebuie să realizeze operația specificată asupra unui polinom individual, chiar dacă operația selectată include două polinoame ca input și nu creează o legătură între cele două (de exemplu: derivarea și integrarea).
- Calculatorul trebuie să afișeze rezultatul calculului în zona de text desemnată.

Cerințe Non-funcționale

- Interfața prin care utilizatorul interacționează cu aplicația trebuie să fie intuitivă și ușor de utilizat.
- Operațiile matematice trebuie să fie implementate eficient.
- Utilizatorul trebuie să poată folosi aplicația fără erori sau buguri neașteptate.
- Calculatorul trebuie să fie implementat corespunzător și să gestioneze corect cazurile limită, erorile sau inputul greșit, fiind pregătit pentru o utilizare necorespunzătoare.

Cazuri de utilizare

Aplicatia va fi utilizată pentru calcularea diferitor operatii asupra polinoamelor.



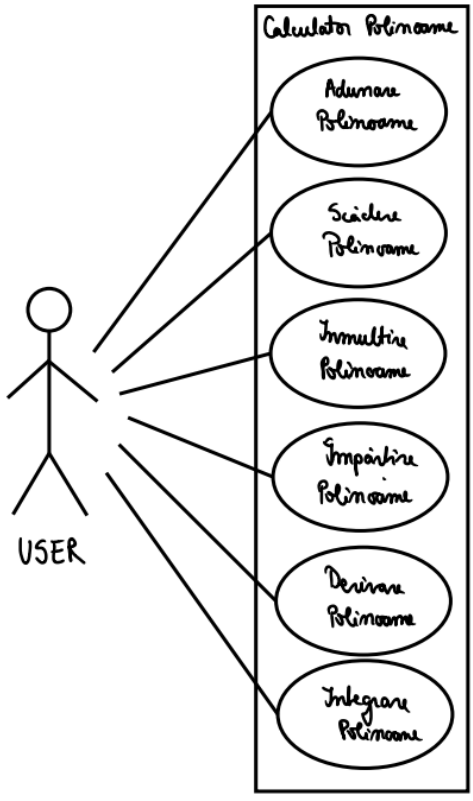
Comportamentul asteptat al aplicatiei

Scenariu principal de succes:

- Utilizare corectă a calculatorului pentru polinoame
    - Actor primar: utilizatorul
    - Utilizatorul introduce 2 polinoame valide și nenule, care respectă formatul specificat.
    - Utilizatorul selectează una dintre operațiile disponibile: adunare, scădere, înmulțire, împărțire, derivare sau integrare.
    - Programul procesează polinoamele.
    - Utilizând metoda specifică operației selectate, calculatorul formează rezultatul care este afișat pe ecran pentru utilizator.

Scenarii alternative:

- Introducerea polinoamelor invalide
    - Utilizatorul introduce polinoame, dar acestea nu sunt valide sau sunt nule. Programul detectează polinoamele invalide și afișează un mesaj de eroare pe ecran, solicitând reintroducerea polinoamelor valide.
    - Se revine la pasul inițial, utilizatorul fiind încurajat să reintroducă polinoamele.
- Selecția unei operații invalide
    - Utilizatorul nu selectează operația, iar aplicatia afișează un mesaj de eroare pe ecran, cerând utilizatorului să selecteze o operație validă.
    - Se revine la pasul inițial.



3. Proiectare

a. Proiectarea OOP

Aplicația este structurată în 3 clase principale, fiecare având un pachet reprezentativ pentru delimitarea claselor care se ocupă cu determinarea modelului de date, care se ocupă de interfață și care se ocupă de logistică și interacțiune între clase.
Clasele au fost definite în așa fel încât să se respecte principiul OOP de abstractizare și encapsulare a datelor. Atributele sunt păstrate private, în special hashmap-ul clasei Polinom, care este accesat în mod limitat din afara clasei.

b. Diagrame UML

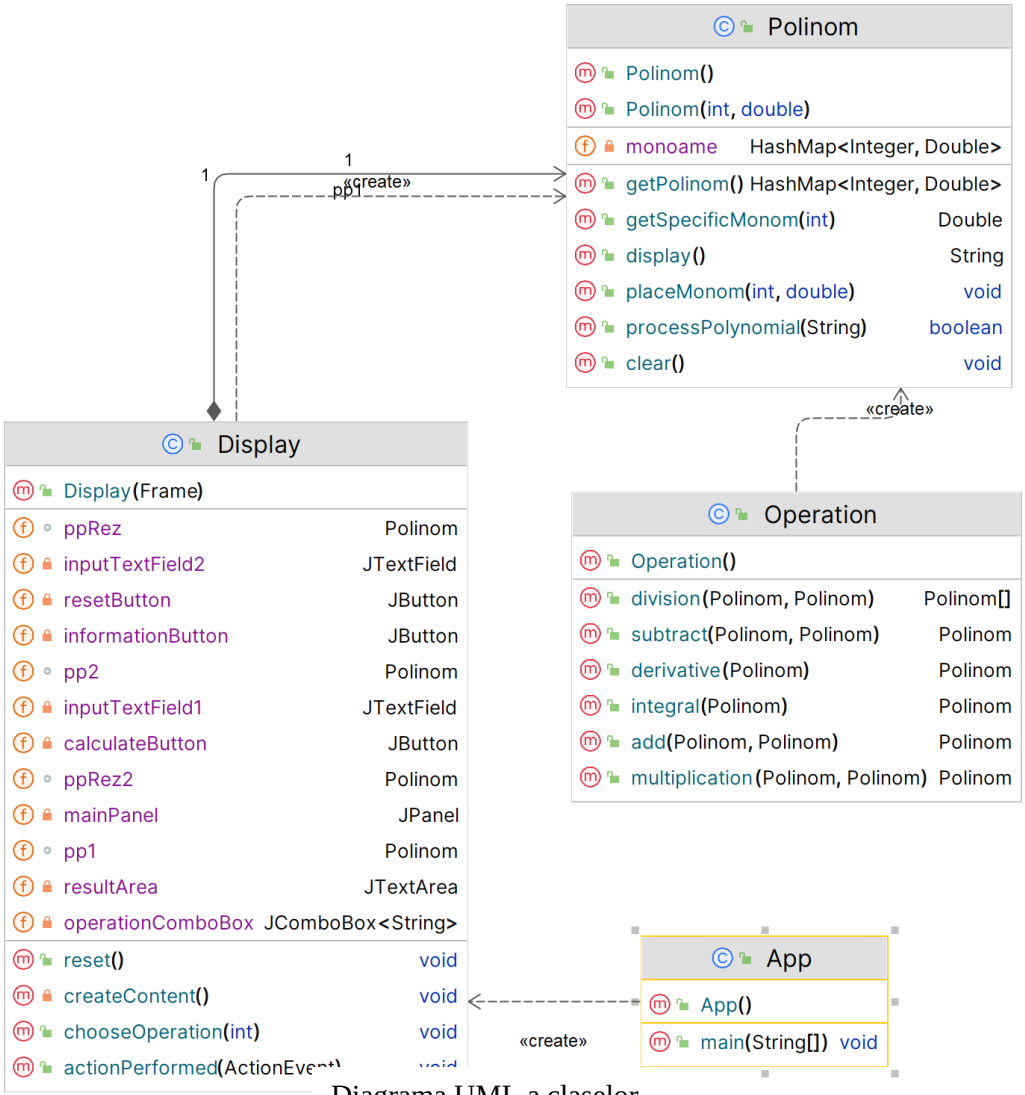


Diagrama UML a claselor

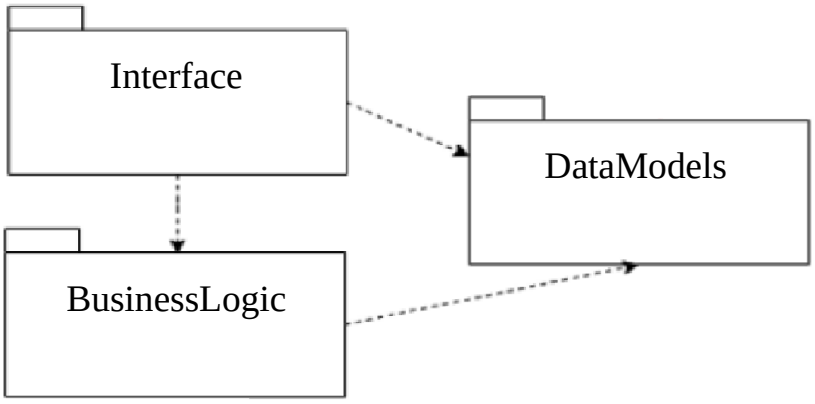


Diagrama UML a pachetelor

c. Structuri de date

Structura principală utilizată este un HashMap pentru stocarea monoamelor pe care le conține un polinom. Acesta este implementat în așa fel încât să nu poată fi accesat decât în mod limitat.

d. Algoritmi folositi

Algoritmii utilizați sunt mai ales cei ce realizează operațiile asupra polinoamelor. Pe lângă aceștia, există și algoritmi pentru conversia unui polinom din text în monoame separate și invers. Acest lucru este implementat cu ajutorul a două regexuri, care selectează secțiuni diferite din șirul întreg și apoi interpretează grupurile. Algoritmul validează formatul și stochează monoamele într-o structură adecvată pentru a permite efectuarea viitoarelor operații.

4.Implementare

În dezvoltarea acestui proiect, am utilizat 3 clase principale, excluzând cele utilizate pentru testare unitară, care vor fi detaliate în capitolul următor.

4.1 Clasa Polinom

Aceasta reprezintă clasa care determină structura principala pe care se defineste proiectul.

Campuri:

`private final HashMap<Integer, Double> monoame;`	O structură de date utilizată pentru memorarea fiecărui polinom este un map de monoame, caracterizat de o pereche cheie-valoare. În această structură, cheia reprezintă exponentul, iar valoarea reprezintă coeficientul unui termen din polinomul respectiv.

Constructori:

`public Polinom()`	Crează un obiect gol de tip polinom.
`public Polinom(int exponent, double coeficient)`	Crează un obiect de tip polinom care are un singur monom, precizat în antetul metodei.

Metode:

`public void placeMonom(int exponent, double coeficient)`	Amplasează un monom nou în polinom.
`public HashMap<Integer, Double> getPolinom()`	Returnează conținutul polinomului pentru a putea fi iterat în afara clasei.
`public String display()`	Asemănătoare unei metode „toString”, returnează conținutul polinomului sub formă de string utilizând un StringBuilder.
`public Double getSpecificMonom(int x)`	Returneaza coeficientul unui monom pe baza exponentului
`public void clear()`	Goleste continutul curent al polinomului.
`public boolean processPolynomial(String polynomial)`	Metoda care se ocupă de transformarea polinomului dintr-un șir de caractere într-un polinom format din monoame, folosind două expresii regulate (unul pentru monoame, unul pentru constanta polinomului).

4.2 Clasa Operation

Această clasă este statică și conține toate operațiile care au loc între polinoamele de forma specificată mai sus. Fiind statică, metodele sunt apelate direct din clasă.

Câmpuri: Clasa Operation nu are atribute, deoarece polinoamele sunt transmise metodelor ca parametri și polinomul rezultat este declarat local în fiecare metodă pentru a elimina nevoia resetării acestuia între operații.

Metode:

`public static Polinom add(Polinom p1, Polinom p2)`	Operația de adunare dintre două polinoame este realizată prin plasarea primului într-un rezultat și apoi iterarea asupra celui de-al doilea, realizând adunarea între coeficienți unde exponentul este identic.
`public static Polinom subtract(Polinom p1, Polinom p2)`	Operația de scădere dintre două polinoame este realizată prin plasarea primului într-un rezultat și apoi iterarea asupra celui de-al doilea, realizând scăderea între coeficienți unde exponentul este identic.
`public static Polinom multiplication(Polinom p1, Polinom p2)`	Realizează operatia de inmultire dintre două polinoame prin iterarea "for in for" peste monoamele ambelor polinoame si realizând: 1. Adunarea exponenților celor 2 monoame inmultite 2. Inmultirea coeficienților celor 2 monoame inmultite 3. Adăugarea noului coeficient rezultat la coeficientul care exista deja pentru acel exponent
`public static Polinom[] division(Polinom p1, Polinom p2)`	Realizează operația de împărțire dintre două polinoame prin iterarea cu un while care este întrerupt cand deîmpărțitul are gradul mai mic decât împărțitorul. Pașii în timpul iterației sunt cei realizați în long division: 1.Împărțirea deîmpărțitului la monomul cu exponent cel mai mare al împărțitorului. 2. Înmulțirea rezultatului cu împartitorul. 3.Scăderea rezultatului din deîmpartit.
`public static Polinom derivative(Polinom p1)`	Realizează operatia de derivare asupra unui singur polinom prin iterarea monoamelor acestuia si realizand: 1. Adăugarea exponentului la coeficient prin înmulțire 2. Scăderea exponentului cu 1.
`public static Polinom integral(Polinom p1)`	Realizează operația de integrare asupra unui singur polinom prin iterarea monoamelor acestuia și realizand operatia inversă derivării: 1. Elimină exponentul din coeficient prin împartirea cu acesta 2. Incrementarea exponentului cu 1.

4.3 Clasa Display

Această clasă realizează crearea interfeței grafice, având de asemenea clase lambda pentru implementarea ActionListener-ilor. Aceasta determină ce se va afișa după realizarea unei operații și ce va vedea utilizatorul când folosește aplicația. Implementarea interfeței grafice a fost realizată utilizând elemente Swing UI, amplasate pe un panou fără manager de aspect pentru a oferi libertate în poziționarea componentelor.

Câmpuri:

`Polinom pp1 = new Polinom();` `Polinom pp2 = new Polinom();`	Polinoame preluate după procesarea inputului din textField.

`Polinom ppRez = new Polinom();`
`Polinom ppRez2 = new Polinom();`
`Polinom[] rez;`
	Polinoame rezultat. Al doilea există pentru cazurile în care există input pentru ambele polinoame iar operația selectată este derivarea sau integrarea. Array-ul de tip polinom este pentru stocarea rezultatului împărțirii sub forma: rez[0] = quotient , rez[1] = remainder

`private final JPanel mainPanel;`	Panelul principal pe care sunt amplasate componentele.

`private JTextField inputTextField1;`
`private JTextField inputTextField2;`
`private JTextArea resultArea;`
	JTextField-uri pentru preluarea inputului, JTextArea pentru afisarea rezultatului, permițând afisarea pe mai multe linii.

`private JComboBox<String> operationComboBox;`	JComboBox pentru selectarea operatiei dintr-un meniu de tipul “drop down”.

`private JButton calculateButton;`
`private JButton resetButton;`
`private JButton informationButton;`
	Primul buton are funcționalitatea de a porni procesul de preluare, verificare și calculare a rezultatului diverselor operații.
	Al doilea buton golește componentele pentru a primi un nou input.
	Al treilea buton este suplimentar, pentru a reaminti formatul acceptat de calculator.

Constructor:

`public Display(Frame parent)`	Creează interfata grafică, apeland diverse metode pentru amplasarea componentelor. Aici există si ActionListener-ele butoanelor mentionate mai sus.

Metode:

`private void createContent()`	Crează componentele grafice manual si sunt determinate proprietățile acestora. Aici are loc si asignarea componentelor panel-ului principal.
`public void reset()`	Metoda apelată in ActionListener-ul butonului de reset, pentru a goli componentele.
`public void chooseOperation(int selectedIndex)`	Metoda utilizează indexul operatiei selectate din meniul JcomboBox-ului intr-un switch care determină ce operatie va fi realizată si apoi este apelată. După procesarea switchului, se realizează transformarea rezultatului intr-un string prin metoda display() al polinoamelor.

4. Rezultate

Există doua clase create pentru testare unitară, respectiv pentru a testa metodele din clasa de operatii si pentru a testa metodele de parsare si logica cu care sunt implementare polinoamele.

Raport de acoperire

Element	Class, %	Method, %	Line, %
all	50% (2/4)	61% (13/21)	45% (112/246)
BusinessLogic	50% (1/2)	85% (6/7)	97% (67/69)
App	0% (0/1)	0% (0/1)	0% (0/1)
Operation	100% (1/1)	100% (6/6)	98% (67/68)
DataModels	100% (1/1)	87% (7/8)	95% (45/47)
Polinom	100% (1/1)	87% (7/8)	95% (45/47)
Interface	0% (0/1)	0% (0/6)	0% (0/130)
Display	0% (0/1)	0% (0/6)	0% (0/130)

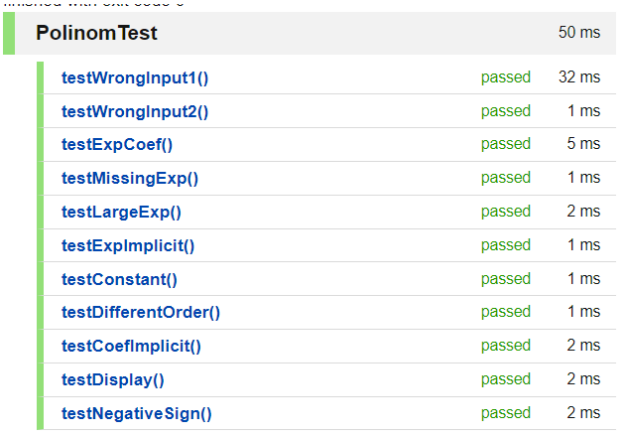
După cum se poate observa, pachetele în care se află cele două clase menționate mai sus au o acoperire de linii mai mare de 90%, fiind testate majoritatea metodelor din cele două clase.

Testarea unitară se realizează pentru toate cele 6 metode de calcul, fiind aplicate diferite inputuri și cazuri limită pentru a testa abilitatea aplicației de a se adapta atât unor scenarii obișnuite cât și unor scenarii neobișnuite.

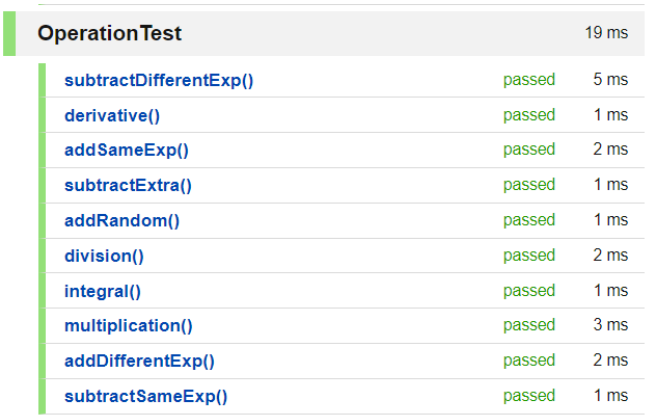
În clasa de Polinom, metodele testate sunt cele de procesare a polinomului și de testare a metodelor care contribuie la încapsularea clasei, respectiv settere și gettere.

Clasa App reprezintă clasa în care se află metoda main, din care are loc doar inițializarea interfeței grafice, motiv pentru care nu este realizată testarea unitară asupra acesteia. Interfața grafică nu este testată cu JUnit deoarece aceasta doar creează legătura cu cele două clase testate, respectiv Operation și Polinom (care sunt testate).

Rezultate teste



PolinomTest		50 ms
testWrongInput1()	passed	32 ms
testWrongInput2()	passed	1 ms
testExpCoef()	passed	5 ms
testMissingExp()	passed	1 ms
testLargeExp()	passed	2 ms
testExpImplicit()	passed	1 ms
testConstant()	passed	1 ms
testDifferentOrder()	passed	1 ms
testCoefImplicit()	passed	2 ms
testDisplay()	passed	2 ms
testNegativeSign()	passed	2 ms



OperationTest		19 ms
subtractDifferentExp()	passed	5 ms
derivative()	passed	1 ms
addSameExp()	passed	2 ms
subtractExtra()	passed	1 ms
addRandom()	passed	1 ms
division()	passed	2 ms
integral()	passed	1 ms
multiplication()	passed	3 ms
addDifferentExp()	passed	2 ms
subtractSameExp()	passed	1 ms

După cum se poate observa, cele 21 de teste create pentru a determina corectitudinea metodelor pe diferite cazuri au trecut cu succes, oferind rezultatul asteptat. Astfel se poate constata corectitudinea algoritmilor pe cazurile testate. Singurul caz descoperit este cel in formatul neasteptat, deoarece metoda de parsare verifică doar impotriva unui sir care nu respecta o structura de polinom, nu erori de tastare.

5. Concluzii

Acest proiect a reprezentat o oportunitate relevantă în dezvoltarea propriilor cunoștințe și abilități și în aprofundarea acestora. Am dobândit abilitatea de a lucra cu structura de date HashMap, am descoperit modalitatea de a transforma un șir prin expresii regulate și am repetat operațiile pe polinoame din perspectiva unui algoritm și nu doar pe hârtie. Proiectul mi-a oferit oportunitatea de a aprofunda și repeta diferite elemente și paradigme ale limbajului de programare orientat pe obiect.

În ceea ce privește dezvoltările viitoare ale aplicației, se poate îmbunătăți metoda de transformare a șirului polinom în monoame cu ajutorul unui regex pentru a captura erori în tastare și alte inputuri greșite care nu sunt acoperite în prezent. De asemenea, se pot implementa mai multe teste cu JUnit pentru asigurarea stabilității și calității aplicației într-o gamă mai largă de cazuri.

6. Bibliografie

1. *Git Lab - https://www.jetbrains.com/help/idea/gitlab.html*
2. *Regex for polynomial expression - https://stackoverflow.com/questions/36490757/regex-for-polynomial-expression*
3. *Regular expressions 101 - https://regex101.com/*
4. *Polynomials – Long divison - https://www.youtube.com/watch?v=8lT00iLntFc*
5. *Java Unit Testing with Junit - https://www.youtube.com/watch?v=vZm0lHciFsQ&t=790s&ab_channel=CodingwithJohn*
6. *Java TreeMap vs HashMap - https://www.baeldung.com/java-treemap-vs-hashmap*
7. *StringBuilder Class in Java with Examples - https://www.geeksforgeeks.org/stringbuilder-class-in-java-with-examples/*
8. *Resurse Laborator - https://dsrl.eu/courses/pt/materials/PT_2024_A1_S1.pdf*