DOCUMENTAȚIE PROIECT

Calculator Polinomial

Stancu Mihai-Cristian

1. Obiectivul temei

a. Obiectiv principal

i. De a crea un calculator polinomial

b. Obiective Secundare

- Analizarea problemei și identificarea cerințelor (Capitolul 2)
- Proiectarea calculatorului polinomial (Capitolul 4)
  - descrierea tuturor pachetelor
  - descrierea tuturor claselor
- Implementarea calculatorului polinomial (Capitolul 3)
- Testarea unitara a aplicației si a functionalitatii(Capitolul 5)

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

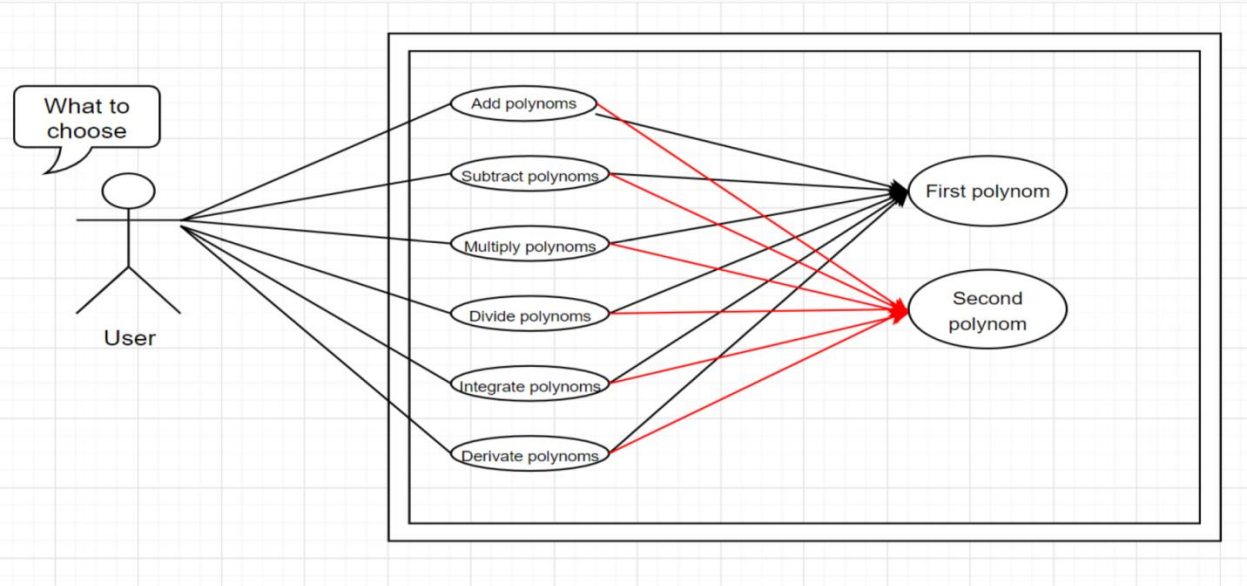
2.1 Cerințele funcționale si non-funcționale

Dupa ce am citit problema si am analizat textul, pot indentifica principalele cerinte:

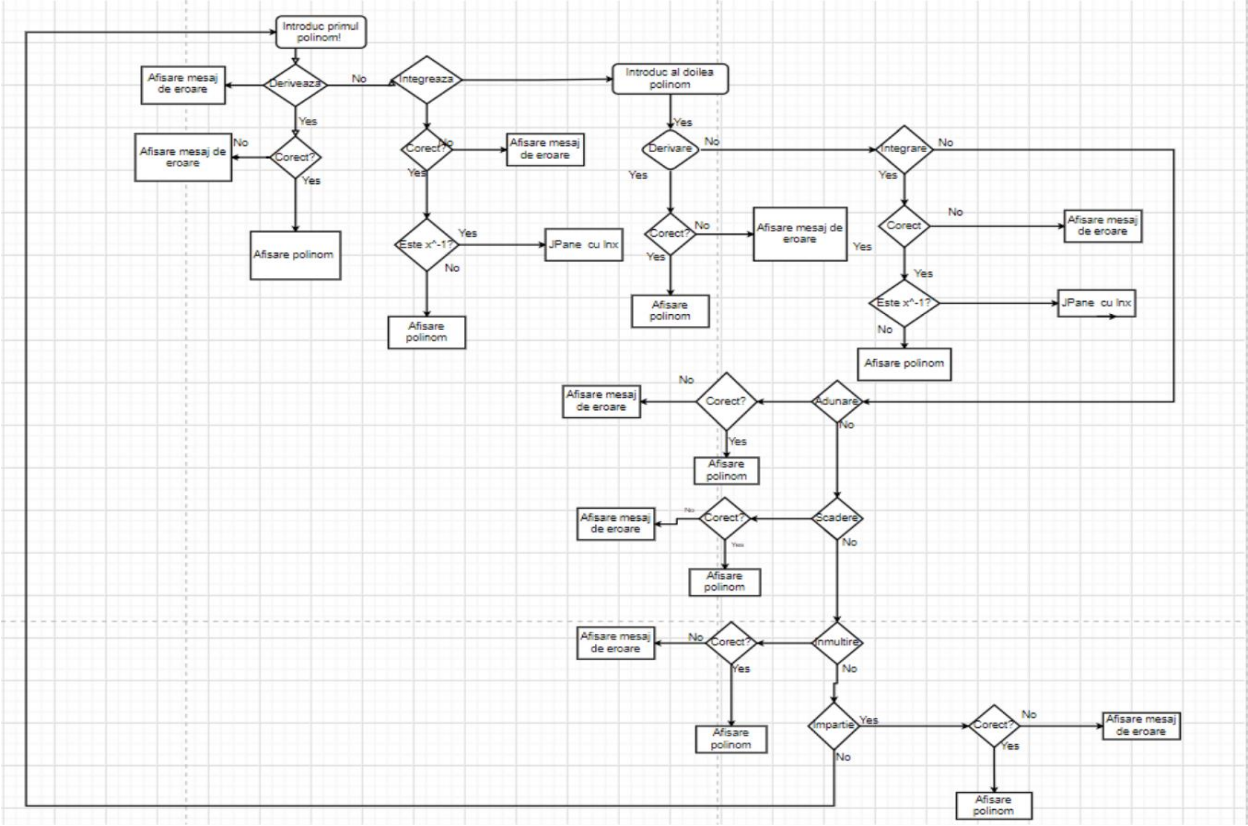
1. functionale pentru calculatorul polinomial:
   - Trebuie sa am campuri de text in care sa scriu polinoame(JTextField)
   - Trebuie sa pot cumva sa creez o legatura intre TextField si operatii
     - La integrare si derivare folosesc doar 1 TextField
   - Trebuie sa am un Combox sau ceva ScrollBar in care sa pot alege operatia

2. Cerințele non-funcționale ale calculatorului care sunt logice cand vine vorba despre ajutarea utilizatorului:
   - Trebuie sa fie intuitive
   - Trebuie sa aibe validari
     - Impartire la 0
     - Scriere gresita
     - Cazuri speciale

2.2 Diagramă use-case



2.3 Descrierea use-case-urilor



3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

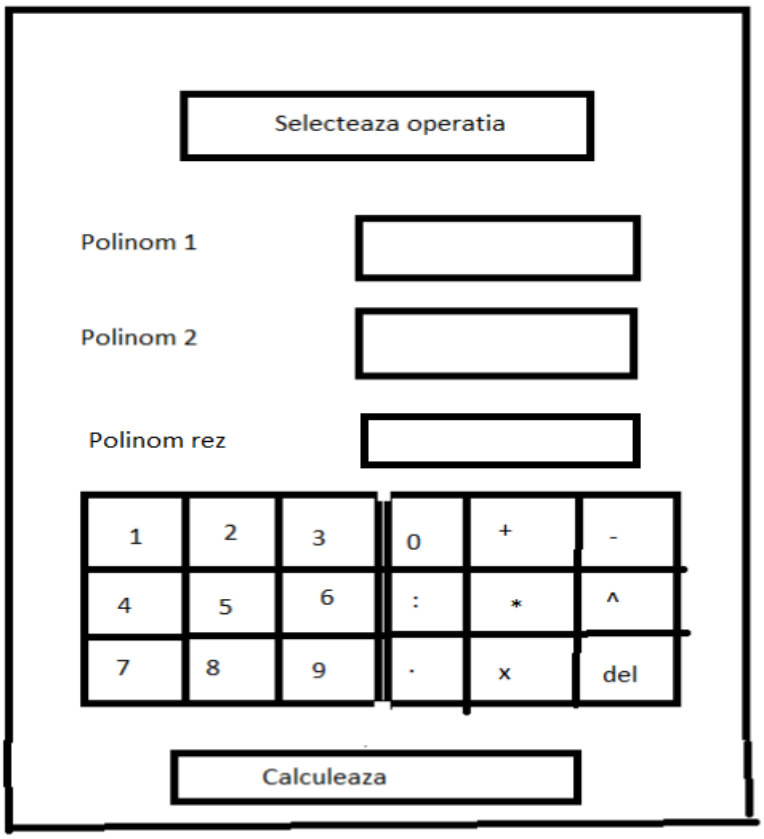
3.1 Decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete

In etapa 3, dupa ce am analizat celelealte 2 etape ce parte functionala si non-functionala, mi-am creat un mock in Paint pentru a incerca sa vizualizez partea de interfata grafica si sa incerc s-o duc in arhitectura MVC pentru a avea o structura clasa si concisa pentru program.

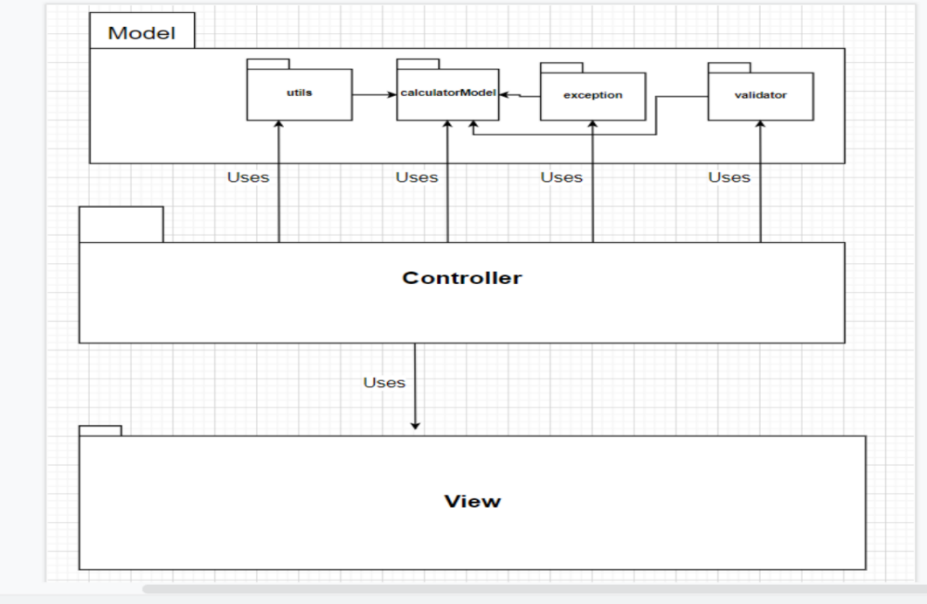
Am 3 pachete, Model,Controller si View:

- *Modelul* contine partea de calcul a aplicatiei si toate validarile, practic toata partea de logica care s-ar gasi intr-un packade de service la aplicatie pe scara mare. Am clasele de Monomial,Polynomial,Operation si PolyCalcModel. In partea de validari am clasas PolynomValidator care imi valideaza inputul si mai am si exceptiile si utilitatile in clase aferente.
- *Viewul* contine partea de interfata grafica cu in clasa PolyCalcView unde am toate butoanele si arhitectura descriptiva aplicatiei.
- *Controllerul* contine comunicarea dintre View si Model si are toti Listenerii si legaturile dintre utilizator si aplicatie

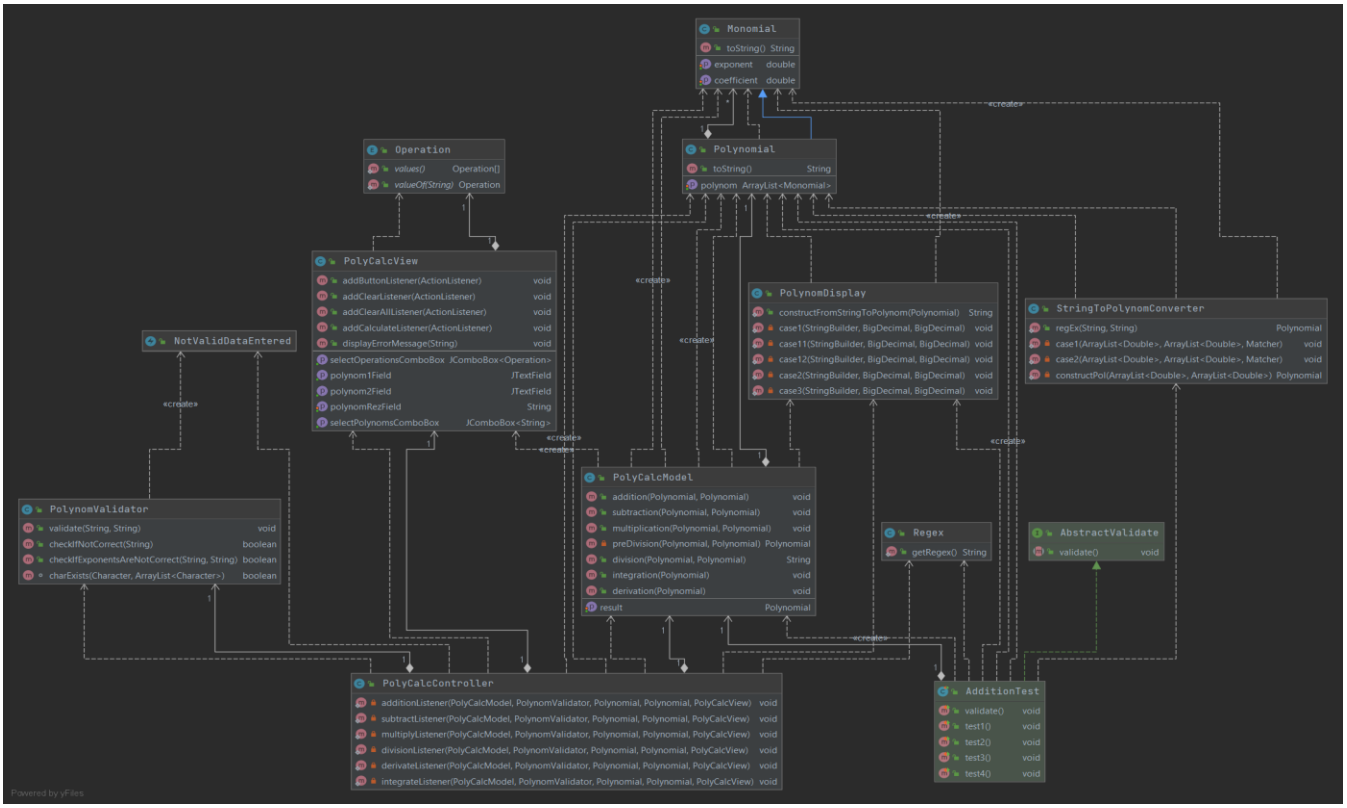
Mock



3.2 Diagrama de pachete



3.3 Diagrama de clase



4. Implementare

La implementarea temei am inceput prin analizarea a detaliilor matematice despre polinoame, si am incercat sa imi creez un sistem de date de intrare si de iesire conform cerintelor temei. Mi-am creat prima oara clasele de *Monomial* si de *Polynomial* si de acolo am inceput sa dezvolt.

1) PACKAGE CONTROLLER

- In clasa ***PolyCalcController*** care are partea de comunicare dintre model view si validator si pe care o instantiez prin intermediul unui contructor care primeste ca parametrii atributele clasei, anume modelul(PolyCalcModel), view-ul(PolyCalcView) si validatorul(PolynomValidator). De asemenea, in interiorul constructorului am si listenerii, anume ButtonListener, ClearListener,CalculateListener si ClearAllListener.
- In clasa ***ButtonListener*** care are logica de alegere a butoanelor. Verific prin intermediul logicii conditionale ce camp din ComboBox este ales pentru a alege in ce JTextfield scriu(contine 2 butoane, polynom1 si polynom2). Pentru ambele fielduri din ComboBox, verific daca apas pe tasta DEL sau nu. In cazul tastei DEL, imi setez textul din textfield ca vechiul next fara ultimul caracter, iar in celelate cazuri de cifre si operanzi, doar dau append la textul din textfieldul ales cu caracterul pe care l-am ales. In caz de apare o eroare la alegerea butonului, generez o exceptie de tipul NotValidDataEntered pe care mi-am creat-o eu in packageul de exceptii.

- In clasa ***ClearListener*** imi aleg ce JTextField vreau sa sterg. In view i-am dat fiecarui buton de Clear un id pe care il verific in ClearListener. Am ales sa fa casa pentru a nu avea in interata indice la fiecare buton de Clear. Dupa ce aplic logica conditionala pe id-ul pe care l-am pus pentru fiecare buton, in momentul in care apas pe unul dintre cele 3 butoane de clear, aplicatia va stii pe care dintre butoane apas si imi va seta textful din textfield la NULL.

- In clasa ***ClearAllListener*** setez toate cele 3 JTextFielduri la null.

- In clasa ***CalculateListener*** am listenerii pentru alegerea operatiei din JComboBoxul “selectOperationComboBox”. In prima parte a clasei, imi creez polinoamele de tipul clasei Polyomial prin intermediul metodei din packageul UTILS “StringToPolynomConverter”, care imi ia textul din TextField si mi-l transforma in clasa polinom, iar apoi imi validez inputurile din JTextField al ambelor polinoame. Apoi , prin intermediul unui switch, verific pe ce buton am apasat pe JComboBoxul operatiilor, si cf. acelui buton ,efectuez operatia pe inputuri, iar rezultatul, dupa ce este il validez si pe el , il introduce in TextFieldul result. Pentru fiecare operatie in parte am metodele declarate sub aceasta clasa, iar pentru input incorect, arunc exceptii de tip NotValidDataEntered.

2) PACKAGE EXCEPTIONS

- In clasa ***NotValidDataEntered*** am o exceptie ce extinde RuntimeException si pe care o folosesc in intreg proiectul.

3) PACKAGE MODEL

- In clasa ***Monomial*** am campurile exponent si coefficient de tip double, 2 constuctori(unul gol si unul cu atributele), getteri si setteri pentru ambele campuri si metoda predefinita toString. ***Am ales sa fac monom***

cu atribute double, deoarece voi avea polinoamele extrem de maleabile, pot pune atat coeficienti si exponenti double, cat si int pe acelasi input. Diferentierea o fac in momentul afisarii, care imi va scoate acele zeroui de dupa virgule.

- In clasa ***Polynomial*** am o lista private de monoame si un constructor. Am de asemenea getteri si setter pentru ArrayListul de monoame cat si o metoda toString.

- In clasa ***Operation*** am un enum al tipurilor de operatii pe care il folosesc in VIEW.

- In clasa ***PolyCalcModel*** am toate cele 6 operatii pe polinoame. ***Am cate o metoda pentru fiecare si cateva metode ajutatoare pe care le-am facut separate pentru a respecta limita de 30 de linii pe metoda.***

4) PACKAGE UTILS

- In clasa ***Regex***, doar imi returnez regexul pe care il dau de la tastatura intr-o variabila static finala regex .

- In clasa ***StringToPolynomConverter,*** convertesc inputul din JTextField in clasa Polynomial. Aici am 2 cazuri mari ,cand grupul 4 din regex e NULL(nu am exponent) si nu e NULL. Cand este NULL , am alte 2 cazuri, cand grupul 3 e NULL(adica am constante) si cand nu este NULL. Pentru toate cazurile,verific daca grupul 1(adica semnul) este gol, deoarece mi-am setat ca pentru numere positive sa nu mi apara nimic, iar pentru cele negative sa mi apara minus. De asemenea, pentru fiecare verific daca in grupul 2(adica coeficientul) este gol sau nu(daca este gol, inseamna ca este 1 acolo, iar pentru elementele in X, este ciudat sa apara acel 1 in fata).

    - ***Cu metoda regEx*** imi sparg cu ajutorul regexului stringul si imi extrag coeficientii, semnele si exponentii. In metoda ***constructPol*** imi asamblez Polinomul prin adaugarea succesiva in rezultat de coeficienti si exponenti formati in ArrayListurile din metoda principala.

- In clasa **PolynomDisplay** am metoda principal "***constructFromPolynomToString***" care imi transforma polinomul creat la metoda anterioara in string pe care sa-l pot pune in TextFieldul rezultatului. ***Aici am 3 mari cazuri*** pe care le sparg in metode separate pentru a respecta acea limita de 30 de randuri pe metoda.
    - ***Cazul 1*** imi verifica daca exponentul nu este 0 si daca coeficientul nu este 1. Aici am de asemenea 2 cazuri, anume daca valorea exponentului este 1 sau nu, cazuri pe care le-am spart de asemenea in alte 2 metode pentru a mentine acele 30 de randuri in frau. Pentru fiecare caz verific daca am intValue==doubleValue, si am ramuri separate pentru fiecare. Am folosit StringBuilder pentru constuire, deoarece este mai rapid si nu creaza mereu alta zona de memorie, ci doar adauga la finalul stringului.
    - ***Cazul 2*** verifica daca exponentul nu este 0 si coeficientul e 1. Si aici verific daca intValue==doubleValue pentru coeficient si pentru exponent, evident daca imi permite ramura.
    - ***Cazul 3*** este pentru exponent 0 si coeficientul oricat, practic este pentru constant, si aici de asemenea verific intValue==doubleValue.

- CAZURI SPECIALE LUATE IN CALCUL LA METODA ***PolynomDisplay***:
    1) Sa nu am -+ sau +-
    2) Sa nu am 1 in fata la X
    3) Sa nu am X^2.00
    4) Sa nu am acel + in fata la polinom
    5) Sa nu am 3.33333, ci doar 3.33, am folosit doar 2 zecimale de dupa virgula

5) PACKAGE VALIDATOR

- Am clasa ***PolynomValidator*** care contine toata logica de validare
  - Aici am metoda principiala ***validate*** imi arunca exceptie de tip NotValidDataEntered daca nu trece testele.
  - Metoda booleana ***checkIfNotCorrect*** imi verifica sa nu am 2 semne de ++ sau de – unul dupa altul.
  - Metoda booleana ***checkIfExponentsAreNotCorrect*** imi verifica daca am exponentii in ordine corecta, daca nu, arunc o exceptie si afisez in Controller un JOptionPane cu “Exponents are unordered”. In metoda imi adaug toti coeficientii intr-un ArrayList<Double> si dupa i ordonez si verific daca nu lista ordonata si cea initiala nu sunt la fel, returnez true(adica exp nu sunt corect pusi),altfel returnez 0. ***Tot aici verific sa nu pun nr cu acelasi exponent unul dupa celalalt pentru a nu avea X-X la polinom 2, anulez impartirea cu 0 practic***

6) PACKAGE VIEW

- Impartire:
  - Partea declarativa abstracta a tuturor atributelor, sunt facute private pentru a spori abstractizarea aplicatiei.
  - ***PolyCalcView*** care contine intreaga asezare pe paneluri a butoanelor si a JTextFieldurilor. TextFieldurile sunt non-editabile pentru a nu lasa utilizatorul sa tasteze si doar sa folosesca butoanele
    - mi-a usurat un pic munca pe partea de validari, doarece nu am mai tinut cont de spatii si de alte lucruri exterioare;
  - Partea de metode contine adaugarea butoanelor la listeneri pentru a fi creati in Controller, si de asemenea contine si getteri si setteri pentru textul din TextField unde se afla polioamele

5. Rezultate

Pentru testare am 2 packageuri, ***testOperations*** si ***testMethods.***

- In packageul ***testOperations,*** verific corectitudinea fiecarei operatii. Am 10 inputuri si imi aleg 4 din ele pe care sa testez. Am o metoda are ca imi face toate cele 4 teste de cate 5 ori (@RepeatedTest). In fiecare test imi creez 2 polinoame prin metoda ***StringToPolynomConverter.*** Apoi imi construiesc rezultatul cu metoda calculului din ***PolyCalcView.*** Imi fac stringul aferent polinomului prin metoda ***PolynomDisplay,*** iar cu ”Assertions.assertEquals(str, actual);” imi testez daca operatia respectiva este corecta.

- In packageul ***testMethods,*** verific corectitudinea metodelor ***StringToPolynomConverter*** si ***PolynomDisplay.*** La ***StringToPolynomConverter*** imi iau un input pe care il transform in polinom prin metoda aceasta, adaug coeficientii si exponentii in niste ArrayList<Double>, si fac assertEquals intre valoarea asteptata a coef si exp si valorea lor din aceste liste.

- Observatii:

1.) Pentru X^-1 la integrare mi-am pus sa imi dea un JOptionPane cu “the result is ln(x)” ,dar in TextField la result am pus 0, deci la testare trebuie sa pun 0.
2.) Am testat si pe exponenti negativi si pe pozitivi si pe double si toate imbinate si am rezultate corecte.
3.) Pentru imparitea la 0 la sa X-X merge totul bine.(am facut verificari pentru asta in validator)
4.) Singura problema este ca am doar 2 zecimale dupa virgula, asa ca la integrarea lui X^1000 o sa am 0 la coeficient
    a. se poate schimba usor din clasa ***PolynomDisplay*** cu adaugarea inca a unei cifre dupa virgula.

5.) Operatiile de ***PolynomDisplay, StringToPolynomConverter*** merg pe aproape orice inputuri ,asta datorita cazurilor multiple luate in metode.
    a. Doar daca de exemplu pun “X+”, o sa imi vada ca si ”X+1”, dar la “X++” imi da eroare potrivit validarii pe care o fa

6. Concluzii

1) Dezvoltari :
    a. sa imi verifice corectitudinea textului dinamic,adica in timp ce -l scriu
    b. validarea la polinom trebuie reparata pentru unele cazuri
    c. designul
    d. implementare JavaFx la urmatoarea tema.
    e. o metoda de evalualuare a polinomului (asta este destul de usor).

2) Ce am invatat:
    a. am invatat SWING
    b. am invatat MVC
    c. mi-am creat o idee asupra urmatoarelor proiecte
    d. refactoring

3) Concluzii:
    a. la proiectul urmator ar trebui sa imi fac prima oara arhitectura si sa stau mai mult sa gandesc de dinainte codul, iar abia dupa ce am gandit totul, sa incep sa implementez Este mult mai rapid si este o buna practica pentru viitor, deoarece asa imi pot tine metodele mai succinte si codul mai curat.

7. Bibliografie

1) https://www.udemy.com/course/java-se-programming/
2) https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html
3) https://www.youtube.com/watch?v=dTVVa2gfht8 (Derek Banas MVC)
4) https://www.youtube.com/watch?v=uUzRMOCBorg&t=682s (Git Tutorial)