Tema 1

Calculator pentru operatii cu polinoame

UTCN, Orsan Tudor, Grupa 30225

1. Obiectivul temei:

Obiectivul principal al acestei teme este de a face o interfata grafica ce sa interactioneze cu utilizatorul, si prin aceasta interfata sa poata introduce polinoame (polinomul 1 si polinomul 2), dupa care sa aleaga ce operatii doreste sa faca pe aceste polinoame. Dupa ce se executa operatia, sa se afiseze rezultatul ei pe interfata.

Obiective principale : Realizare GUI, Operatia de adunare, scadere (corecte), afisare corecta pe GUI.

Alte obiective ce pot fi considerate principale, insa le voi considera secundare: Operatiile de inmultire, impartire, derivare, integrare (vorbit la (4) implementare), testare cu Junit (vorbit la (5) testare), folosire model MVC pentru interfata (vorbit la (3) si (4)), folosire pattern matching pentru a gasi coeficientii si gradele polinoamelor date ca parametrii (folosind functii din pachetul Regex) (vorbit la (3) si (4)) si altele.

Pentru a se ajunge sa avem implementate GUI si operatiile, putem folosi modelul MVC (vorbit (3) si (4)), care face o interfata a utilizatorului bine organizata pe portiuni (logica, vederea si controlul), putem folosi pattern matching pentru a usura munca de gasire a variabilelor cu care lucram (coeficienti si grade), si testarea cu Junit ne poate oferi siguranta ca acest cod functioneaza cum trebuie (pe cazuri speciale sau obisnuite) . Cele 4 operatii descrise la obiective secundare nu ajuta la indeplinirea scopului principal, dar adunarea si scaderea sunt considerate a fi prioritare.

1. Analiza problemei / cazurile de utilizare:

Acest calculator de polinoame are anumite cazuri de utilizare, si aceste cazuri pot fi descrise cu ajutorul use-caseurilor.

Cazurile de functionare ale programului sunt acestea:

- In primul rand, ar trebui sa poata fi inserate polinoame (pentru ca facem operatii intre polinoame, se pot insera 2 polinoame, iar al 3-lea „text field” va fi folosit pentru afisarea rezultatului dintre cele 2 polinoame, sau daca se face operatie doar pe 1 polinom ca la derivare folosim doar primul polinom;

- Dupa, selectarea operatiei matematice se face prin butoane, exista un buton pentru fiecare din cele 6 operatii;

- Aceste 6 operatii sunt: adunarea celor 2 polinoame, scaderea lor (din primul scadem al doilea), inmultirea lor, primul il divide pe al doilea, la derivare derivam doar primul polinom introdus, iar integrarea la fel, tot a primului polinom introdus; (este necesar ca in fiecare din cele 2 campuri sa fie scris un polinom pentru ca operatiile sa poata fi executate, o constrangere).

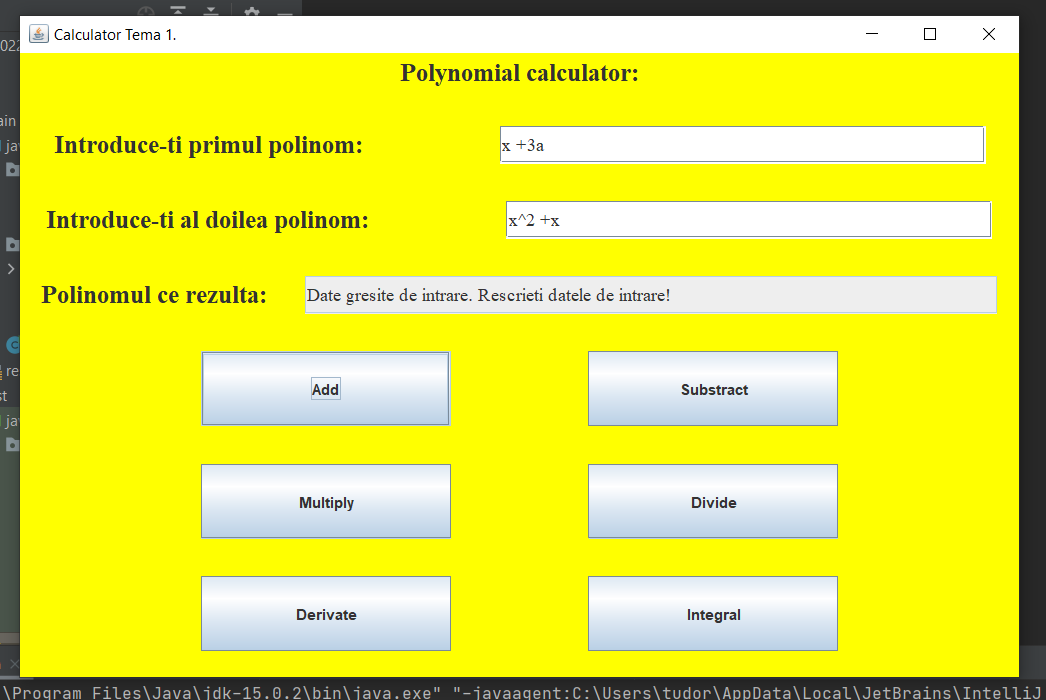
- La final, ar trebui sa se afiseze in acel text field rezultatul dupa operatie, si acesta sa fie rezultatul final al programului (si cel dorit);

- Daca se doreste scrierea altor polinoame, se pot modifica cei 2 parametrii, adica cele 2 polinoame de inceput, si sa se aleaga alta operatie pentru alt rezultat. (In caz de eroare va exista si mesaj pentru eroarea respectiva);

Cerintele non-functionale ale programului sunt:

-Ar trebui sa fie usor de folosit de catre utilizator, sa fie clar unde trebuie sa introduca date, cum face operatiile si unde se afiseaza rezultatul.

-In caz de erori, sa fie evidentiat faptul ca avem acea eroare, si sa cunoasca utilizatorul faptul ca trebuie sa schimbe forma polinoamelor daca este nevoie (a intrarilor adica).



-Sa poata fi reutilizabil, daca se doreste operatie intre alte 2 polinoame, sa se poata face direct, fara multi pasi in plus.

Pentru aceste use-caseuri, voi lua pe rand fiecare si voi evidentia flow-chartul acelui use case:

Use case 1: Add polinoame;

Primary actor : User;

Scenariul de succes principal:

* Se insereaza 2 polinoame in gui, userul alege operatia de adunare, dupa apasarea butonului se calculeaza rezultatul si se afiseaza pe ecran.

Scena alternativa de functionare:

* Daca se introduc polinoame gresite (cu alte date de intrare decat variabila x si structura unui polinom), atunci se afiseaza la text fieldul rezultatului faptul ca nu sunt bune datele de intare, si acum userul poate introduce date noi de intrare. Sunt multe posibilitati de introducere a datelor, oricare mod care nu are forma polinomului dorita se considera o eroare si nu se executa operatia in acest caz.

Use case 2 : Sub polinoame;

Sunt aceleasi conditii ca pentru adunare, nu se schimba nimic, tot ce trebuie stiut este ca din primul scadem al doilea.

Use case 3 : Div polinoame;

In mare, este identic, dar sunt unele constrangeri in plus pentru divizare: (deci se schimba conditiile la introducerea polinoamelor). Aceasta operatie functioneaza doar pentru cand impartirea celor 2 polinoame este un alt polinom exact, deci fara rest, si in plus, trebuie ca toti coeficientii sa fie intregi, deci nici unul sa nu fie real. In rest merge la fel, mai multe despre aceasta operatie voi vorbi la implementare (capitol mai jos).

Use case 4 : Impartire polinoame;

Identic cu use case-ul de la adunare. (nimic diferit dpdv al folosirii)

Use case 5 : Derivare polinom;

Din nou se schimba o singura conditie, cea de la scrierea polinoamelor la inceput: Pentru fuctionare, trebuie scris in ambele polinoame cate un polinom, iar cel care este derivat si afisat la rezultat este primul polinom.

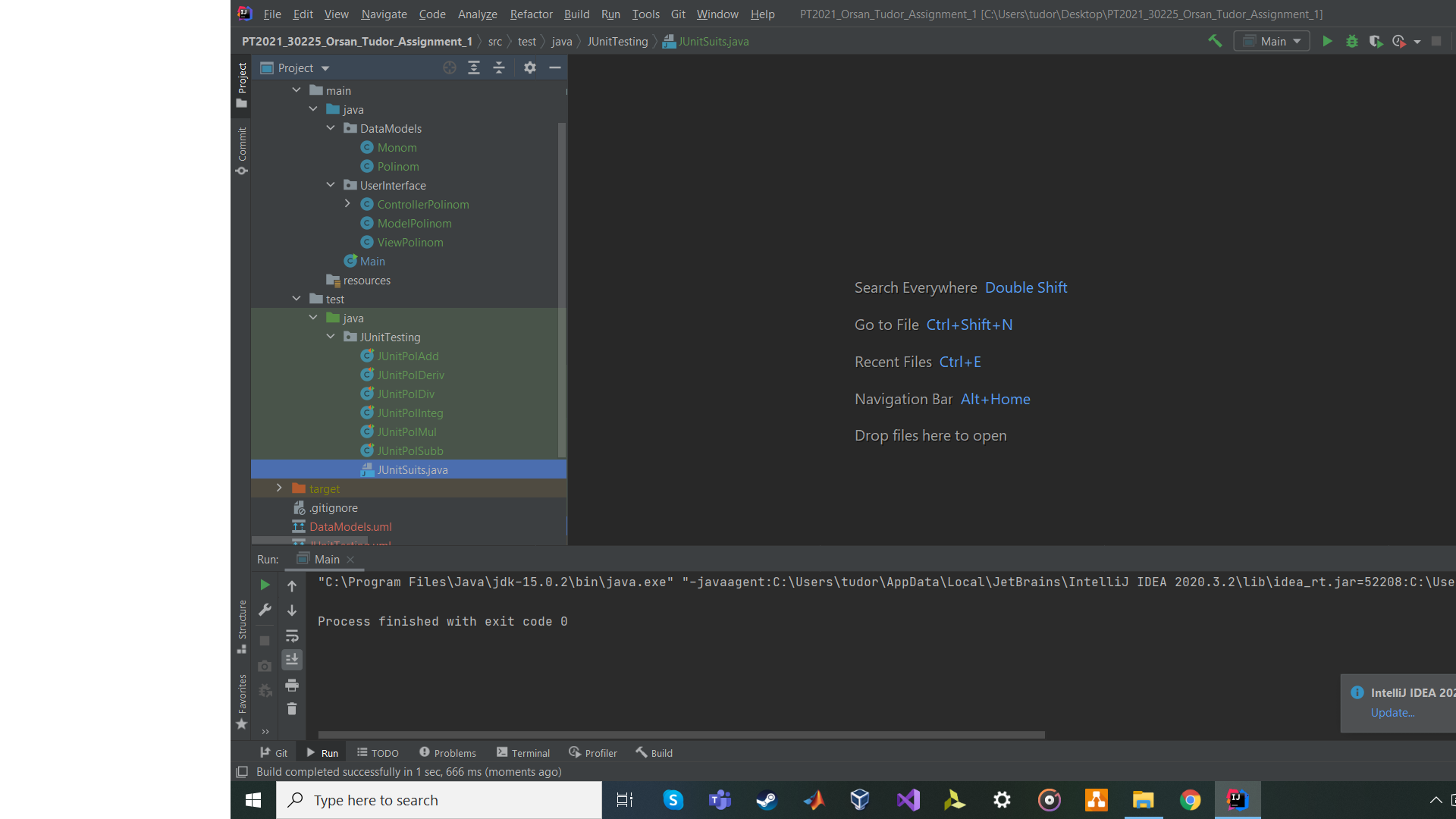
Use case 6 : Integrare polinom;

Se face identic cu derivarea, aceleasi conditii. Intrarile coeficientilor tot trebuie sa fie intregi, insa aceasta este singura operatie unde se afiseaza numere in real la fieldul rezultatului. (dar datele de intrare sunt tot in intreg)

1. Proiectarea problemei:

Pentru proiectarea problemei, am sa intru in mai multe aspecte:

Primul aspect: Pachete



Deoarece sunt nivele diferite de functionare, am impartit in mai multe pachete problema. Partea de logica, si anume clasele ce se ocupa de calculele cu polinoame si forma de polinom (monoame) este pusa in pachetul DataModels; Partea de interfata grafica este pusa in alt pachet, acel pachet continand clasele specifice interfetei MVC. In model ma ocup de convertirea polinomului in rezultat si invers (adica din string in sir de monoame si invers) (aici folosind pattern matching), deci folosesc si clasele de polinom si monom.

Pentru testare, am creat un nou pachet, unde am clasele de testare specifice Junit, si acestea folosesc din nou clasele polinoame si monom ca sa testeze functionarea lor corecta.

Al doilea aspect: Fieldurile (Campurile)

Fieldurile sunt private, in toate clasele, pentru a putea exista o incapsulare mai puternica a codului. Pentru toate fieldurile importante si folosite si altundeva decat in clasa proprie, am creat setteri si getteri.

Al treilea aspect: Interfata Grafica:

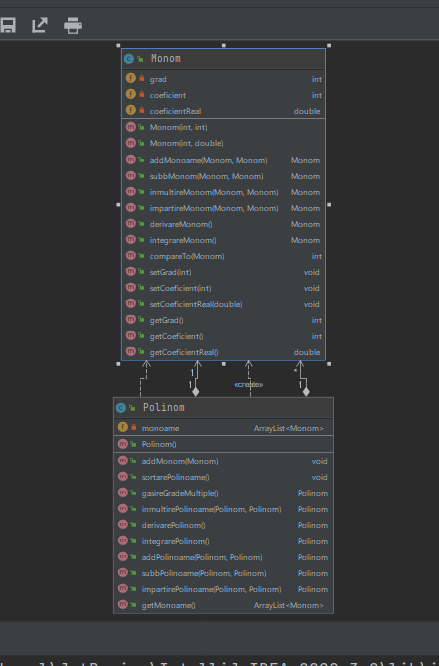
Pentru interfata grafica am ales modelul MVC. Acest model asigura faptul ca interfata este impartita si comunica intr-un mod eficient, si anume modelul face partea de logica, lucreaza cu operatiile ce le folosim pe parcursul programului. View-ul se ocupa de partea vizuala a interfetei, ce vede utilizatorul, adica frameuri, layout-uri, butoane, text fielduri, etc... si aceasta nu intervine cu nimic in logica, insa are o instanta a modelului declarata pentru a folosi datele rezultate din model ca sa le afiseze. Controlerul se ocupa de partea practica, adica are nevoie de instante de la view si de la model pentru a face legatura intre ele. Aici ne ocupam de butoane, si anume la fiecare buton avem o clasa diferita de ActionListener, care atunci cand este apasat un buton, se intampla o actiune, in acest caz se afiseaza pe text fieldul rezultat, rezultatul operatiei. Aceasta componenta este foarte folositoare cand vine vorba de a face legatura intre logica si vizualizarea ei.

Al patrulea aspect: algoritmi

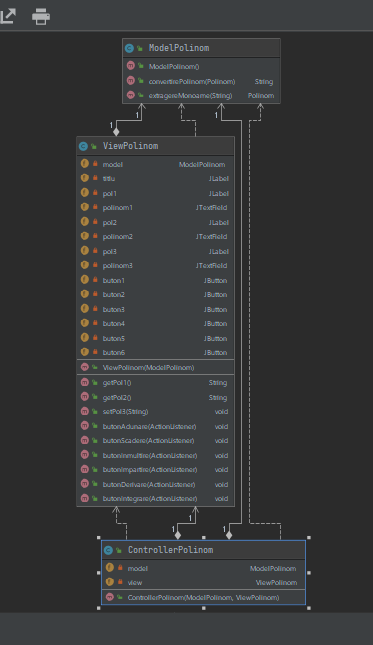
Am folosit un algoritm pentru fiecare operatie, care sa calculeze rezultatul (nu un algoritm predefinit sau prea complex), si am folosit functiile din regex pentru a imparti monoamele intre ele, ca sa pot dupa sa fac operatii mai usor pe ele.

Al cincilea aspect: Diagrama UML:

Data Models:



User Interface:



Am facut diagramele pe pachete (in functie de pachete am facut diagrama de clase), si astfel diagramele principale sunt in pachetul de Data Models si pachetul de User Interface. In special, relatiile principale pe care le am sunt: Am o lista de monoame in polinom, dar pentru ca nu le creez acolo (cu new, nu le instantiez, doar le adaug) ramane relatie de compozitie. In gui, am relatii de dependenta intre cele 3 clase, deoarece se apeleaza una in alta: In controller, avem un model si un view, deci din nou relatie de compozitie, in view avem un model (din nou, neinstantiaz, ci adaugat in constructor) deci iar compozitie, iar in model nu avem nimic deoarece el trebuie sa fie cel mai independent. Vorbesc mai in detaliu la implementare despre cum sunt cele 3 clase, aici am evidentiat relatia lor.

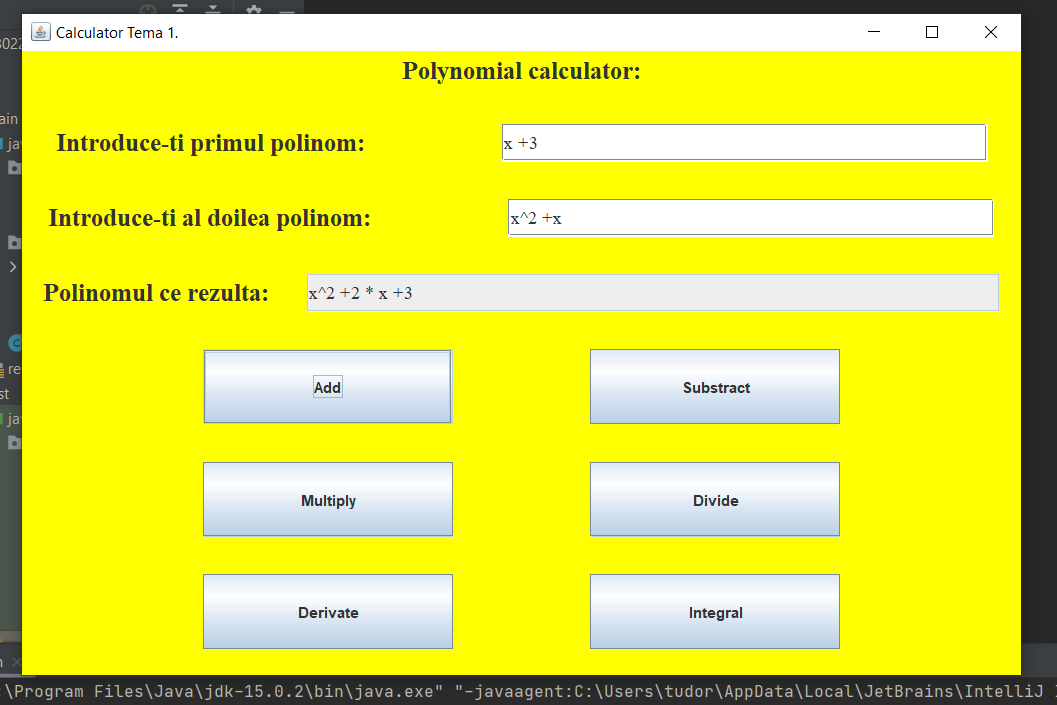
1. Implementarea problemei:

Din cate am zis mai sus, programul este impartit in pachetul de logica, de interfata grafica si de testare. Voi intra in detalii in fiecare pachet, in clasele, metodele si fieldurile folosite:

Pachet data models:

* Am creat 2 clase, monom si polinom. In clasa monom am creat 2 fielduri, si anume gradul unui monom si coeficientul unui monom. Pentru a calcula integrala unui polinom, am adaugat un al 3-lea field, si anume coeficientReal. Prin logica (spre exemplu initializez unul din fielduri intr-un anumit fel pentru a il alege cand doresc), se alege cand sa se foloseasca unul si cant altul (am supraincarcat constructorul monomului, pentru a putea folosi oricare din cele 2 fielduri, atunci cand este nevoie).
* Am facut 6 metode pentru fiecare operatie principala in parte, iar in polinom voi apela aceste metode din monom, creand o relatie de dependenta intre cele 2 clase (pe monoamele din polinom). Logica la aceste 6 operatii este diferita, nu voi intra in detalii prea tare, de exemplu la adunare testez daca polinoamele au acelasi grad, altfel nu se pot aduna, la impartire verific ca polinomul 1 sa aiba grad mai mare, la integrare si derivare verific cazurile in care gradul este 0 sau 1, etc... . Pe langa, mai am o metoda implementata din interfata Comparable, ce ma ajuta sa sortez monoamele in polinom in functie de grad, folositor pentru multe operatii ulterioare. Alte metode sunt doar setteri si getteri.
* In clasa polinoame, am ca field un array list de monoame. Am folosit array list pentru a nu fi interesat de cat de multe monoame introducem, este mai usor de lucrat asa si mai eficient cand programam pe obiecte. Nu initializez lista nicicum (se initializeaza „singura”), doar am o metoda de adaugare a monoamelor, atunci cand creez polinomul. Metoda de sortarePolinom sorteaza cum ziceam in functie de gradul monomului. Metoda gasireGradeMultiple foloseste un vector de frecventa pentru a testa daca in polinomul actual avem sau nu mai multe polinoame de acelasi grad, iar daca avem le „combina” (ca sa avem doar un monom de acelasi grad in polinom).
* Inafara de un getter pe care l-am facut pentru fieldul privat array list de monoame, in rest mai am metodele pentru operatii, le voi explica in mod succint: la logica este mult de explicat, asa ca voi simplifica:
* La fiecare operatie execut gasireGradeMultiple in primul rand pentru a avea doar un grad la monoame in polinom, sa nu se repete 2 grade, iar dupa la adunare: intai adaug cu un double for elementele ce rezulta din adunarea a 2 monoame de acelasi grad din ambele polinoame, dupa care adaug cele ramase din primul polinom, dupa care caut in al doilea gradele care nu le-am luat deja prima data. Puteam face si aici cu vector de frecventa, dar asa si daca este double for O(n^2) ineficient, am parcurs cu for-each fiecare polinom. (deci for pe obiecte).
* La scadere, este acelasi concept ca la adunare, am inmultit cu un -1. La inmultire nu a trebui sa ma ingrijorez de conditii (orice grad se inmulteste cu orice grad), si dupa am eliminat „duplicatele” cu gasireGradeMultiple (doar la inmultire mai pot avea dupa operatie). La impartire a fost cel mai complicat, si dupa aplicarea algoritmului de divizare, salvand in polinoame auxiliare, etc... am ramas cu afisarea doar la polinoamele ce se divid exact si au coeficienti intregi. Algoritmul merge bine, nu se opreste decat dupa 50 de iteratii (o conventie facuta de mine), si in acest caz este un mesaj de eroare, cu cerinta de a se introduce polinoame ce se impart exact, altfel va da acel mesaj (nu se va ajunge la 50 de iteratii decat daca diferenta intre gradele maxime este de 50, si am presupus ca nu voi testa pentru atat, dar se poate schimba usor conventia). Pentru a nu da adresele obiectelor una la alta, si sa incurc legaturile intre polinoame, am salvat un polinom in altul cu un for, adaugand pe rand elementele primului polinom la al doilea. La derivare si integrare a fost cel mai simplu, deoarece nu sunt conditii extra in plus, trebuie doar pe rand derivate si integrate fiecare monom si dupa adaugate in alt polinom. Nu este alta logica in plus aici.
* Avand si comentarii pe cod, partea de model de date se termina aici.

Pachet interfata utilizator:



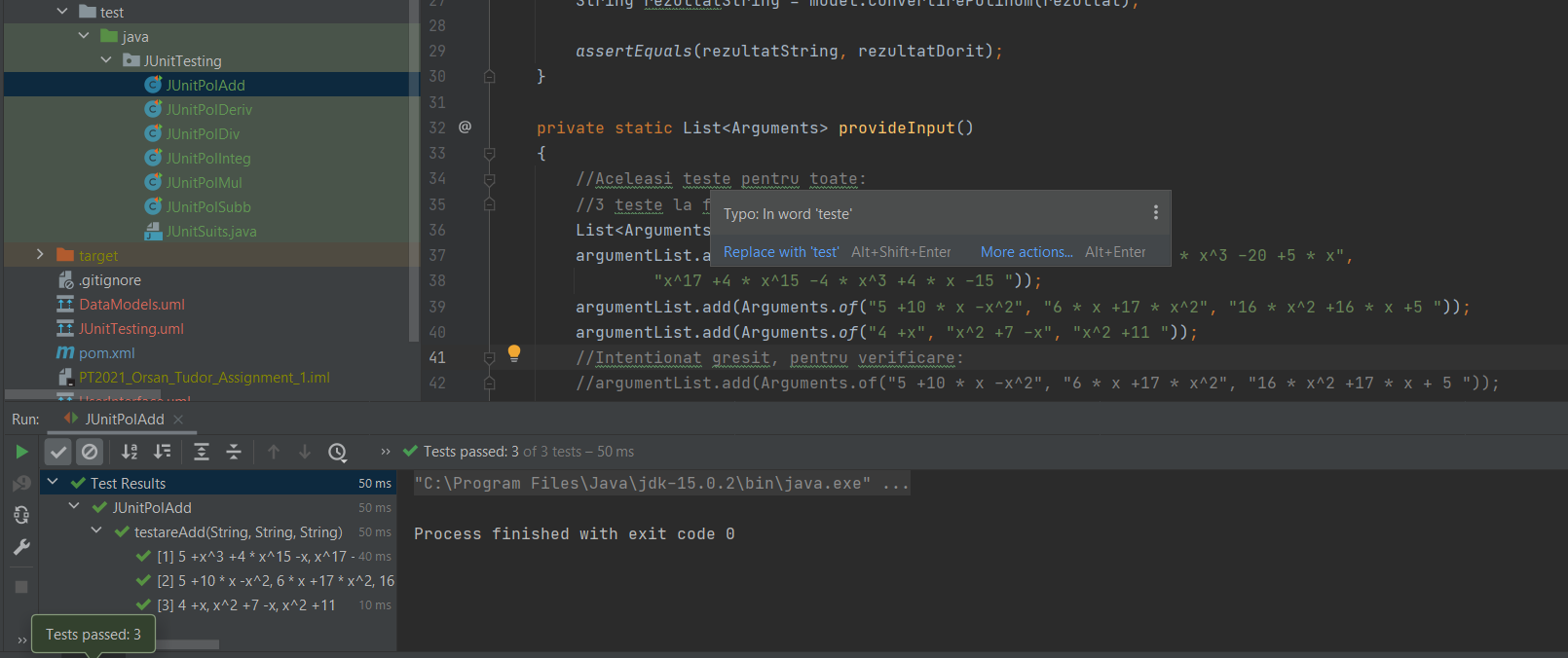
* Am explicat folosinta MVC deja, deci acum voi explica metodele si fieldurile folositoare la cele 3 clase folosite pentru interfata.
* Pentru view am ales 2 layout-uri principale, un box layout pentru a pune toate flow layouturile unul sub altul. Este o abordare simplista, dar functioneaza bine (si are un aspect placut). Am folosit labeluri pentru denumiri, text fielduri pentru introducere si afisare polinoame, butoane pentru a face o operatie anume (din cel e6), si am facut fiecare din acestea la un size potrivit. Am metode de get la cele 2 polinoame si set la rezultat, si pe langa am metode pentru butoane.
* In controler, am salvate butoanele si un view si un model (instantiat). Pentru butoane, apelez metodele din view, cu clasele interne specifice la action listeneri, si astfel in fiecare din clasele interne am metode. Aceste metode se ocupa cu obtinerea polinoamelor din view, convertirea lor din string in polinom cu ajutorul modelului, obtinerea rezultatului dintre cele 2 sau 1 polinoame convertite (operatia specifica metodei din view), si dupa ce convertesc inapoi din sir de monoame in string, pun in view rezultatul la polinomul rezultat. Este acelasi concept la fiecare parte din metode, toate cele 6, iar in plus, mai exista conditie de scriere in text fieldul 3 (cel rezultat) un mesaj de eroare si returnare, pentru a nu incerca sa afiseze un polinom gresit.
* In clasa model, nu am nimic in constructor, nu am fielduri, dar am metode ce implementeaza logica de convertire din string in sir de monoame si din sir de monoame in string. Aceste 2 metode sunt mai lungi de 30 de randuri, insa pentru ca aveam multe conditii, am decis sa le las asa pentru a nu complica si mai tare logica. Pentru convertire din string in monoame, metoda extragereMonoame foloseste metode din regex pentru a face match la anumite patterne. Pentru a usura diferenta intre monoame (si a le grupa cumva mai usor), am gasit diferite match-uri in 3 diferite while-uri, si anume pentru cand am forma „coef \* x^grad” sau „x^grad”, pentru cand am „+-coef”, sau pentru cand am „coef \* x”, adica fara grad. In functie de ce am gasit, pun monoamele respective in polinomul rezultat, de exemplu daca gasest „x”, pun coeficient 1 si grad 1, deci sunt unele cazuri speciale. Logica nu este complicata, insa sunt multe conditii.
* De asemnea, folosesc regex pentru a verifica daca polinomul este de o forma anume, si daca nu are forma ceruta ori nu se afiseaza nimic, ori se afiseaza mesaj de eroare, sunt doar foarte putine cazuri care nu merg cum trebuie, insa m-am focusat pe punctele principale in dezvoltarea acestui program. (si la dezvoltare ulterioara pot fixa aceste mici buguri)
* Pentru convertirea din sir de monoame in string, nu am folosit alte functii, doar am parcurs polinomul care trebuie convertit, si in functie de ce forma are (deci asemanator cu cele 3 conditii de la pattern matching de mai sus), am convertit tot sirul cum trebuie, cu semn potrivit, cu coeficient, grad, etc... Pentru coeficient 0 nu am scris nimic, pentru grad 0 scriu doar coeficientul, pentru grad 1 scriu doar x, si pentru coeficient 1 nu scriu decat x^grad (in caz negativ scriu -1 \* x^grad). Sunt multe conventii facute, si de asemenea am integrat scrierea cu spatii intre coeficienti si grad, de forma „+a \* x^b”, deci din acest punct de vedere este destul de restrictiv.
* Cand vine vorba de coeficient real sau coeficient intreg, am pus conditii in plus in functie de acele 2 fielduri pe care le am in clasa monom. Probabil era mai bine daca faceam o mostenire sau lucram in alt mod, dar am reusit sa fac si asa, chiar daca nu este metoda optima intr-un limbaj pe obiecte (la integrare, am facut operatii speciale cu numere reale, deci aici am avut nevoie de coeficient salvar pe numere reale).

Pachet de testare:

* Aici am o clasa Junit pentru fiecare operatie din cele 6, pentru testare (parametrizata, cu 3 parametrii in aceste cazuri) a unor date de intrare. Voi vorbi despre implementarea acestor teste si rezultatele la urmatorul capitol (5).
* Nu am reusit sa fac Jsuits sa mearga, deoarece am avut probleme cu dependentele, altfel as fi grupat toate 6 clasele astfel, dar voi rezolva pana data viitoare implementarea si astfel, pentru a fi mai eficient.

1. Rezultatele obtinute:

Observatie: pentru input 0, sau cand rezultatul este 0, nu se afiseaza 0, ci nimic, deci semn ca nu au fost bine introduse datele sau ca rezultatul este 0. (Puteam afisa aici 0)



In Junit am testat cate 3 cazuri la fiecare operatie, pentru diversificare. Sunt prea multe cazuri pentru a testa fiecare in parte, dar cazurile alese trateaza cazuri exceptionale de functionare. Pentru toate, inafara de divizare am testat aceleasi 3 seturi de date, iar pentru divizare, pentru ca am nevoie sa se imparta exact si frumos polinoamele, am ales alt set: (in care se impart exact polinoamele, nu intru acolo, ideea ca se impart polinoamele ramane)

Setul 1 : 5 +x^3 +4 \* x^15 -x, x^17 -5 \* x^3 -20 +5 \* x;

In acest exemplu am luat cazul in care avem polinoame de toate formele, cele extrase cu match, x, +5, 4 \* x^15, etc... si am verificat daca le recunoaste si daca le afiseaza cum trebuie. La fiecare a recunoscut bine coeficientul, gradul, semnul, si astfel ca rezultat da bine, afisat corect. (dupa conversie)

Setul 2: 5 +10 \* x -x^2, 6 \* x +17 \* x^2;

In acest exemplu, am luat in mare aceleasi cazuri ca inainte, pentru a testa din nou daca nu sunt alte cazuri exceptionale (la oricare din operatii) care sa nu mearga bine. Nu a dat gresit la niciuna (din teste), si deci acesta a fost testul 2.

Setul 3: 4 +x, x^2 +7 -x;

Am luat un caz mai simplu, pentru a fi sigur ca nu greseste la calcule mici, si da bine si aici (si la derivare, integrare, afiseaza adecvat, de exemplu x derivat da 1). Si pentru constante simple, fara variabila x, da la teste corect, dar am ales sa nu testez si acel caz. (3 cazuri la fiecare mi se par destule)

1. Concluzii:

De la aceasta tema am invatat cum sa lucrez cu pattern matching in primul rand, acest aspect a fost unul pe care nu l-am incercat inainte, de asemenea am invatat cum sa lucrez cu gitlab (dar acest aspect este pe langa tema 1 in sine). Am folosit ce stiam din semestrul trecut la multe din aspectele acestei teme (clase, MVC, implement, ...).

Ca dezvoltari ulterioare, as dori sa fac divizare mai complexa, sa aiba si rest si sa poata fi pe numere reale, sa fac logica sa mearga in toate cazurile, si in cele mai mic (in metodele de convertire si cele de operatii), si ca alte dezvoltari pe langa cele discutate in tema, as putea face alte operatii noi, mai complexe, care sa lucreze cu altfel de numere in monoame, sau sa fac polinoame de mai multe variabile, etc...

In concluzie, au fost multe cazuri speciale de tratat si scopul meu nu a fost sa le tratez pe toate, ci pe majoritatea, dar in rest am putut recapitula conceptele specifice programarii pe obiecte. De acum ma voi focusa sa invat concepte noi si sa le pot aplica in urmatoarele 3 teme.

1. Bibliografie:

* <https://regex101.com/> ;
* <https://mvnrepository.com/> ;
* Materialele de la TP; (Nu sunt prea multe linkuri)