CATALOG VIRTUAL
TEMA/PROIECT PROGRAMARE ORIENTATA PE OBIECTE
DOCUMNENTATIE/README

Tema realizata de: Popescu Maria Teodora, 322CC

- ## Info general:
  - Per total, la aceasta tema am aproximativ 20-24 fișiere .java, clase si interfețe
  - Am pus fiecare clasa in cate un fișier ca sa-mi fie mai ușor la implementat
  - Pentru ca nu mă pricep la engleza, am încurcat notiunile de Firstname si Lastname si am observat aceasta cam prea tarziu ca sa mai modific ceva. Totusi, programul functioneaza in parametrii normali, conform asteptarilor, chiar si asa
  - Tema aceasta mi-a luat cam o saptamana si jumatate, per total. Ca dificultate, nu pare chiar asa de grea, dar e mult de scris. Totusi, am intampinat putina greutate la Sablonul de Proiectare Memento, la Visitor si la Observer, mai ales la Fila Parintelui.
  - Inca nu sunt sigura daca tot ce am implementat este corect, dar conform testelor inventate de mine au functionat.
  - A trebuit sa fac eu niste teste caci: nu stiu sa folosesc fisiere .json, model de date de intrare nu am gasit cand am testat si nu am mai stat sa modific, asa ca m am gandit eu la unele, mai ales ca am imaginație. Asa cum am explicat si in program, la inceputul a ceea ce se va afisa pe consola ca modalitate de verificare, am mentionat nevoia de date de intrare, ca m-am gandit la ceva care m-ar putea ajuta la testarea implementarilor facute de mine.
  - Dovdata functionarii programului consta in ceea ce se afiseaza in consola si functionalitatea interfetei grafice!
  - Datele de intrare sunt inspirate din realitate (profesori, asistenti, chiar daca asign-arile lor la cursuri nu sunt 100% corecte, cel putin la asistenti). Am luat 4 cursuri, 4 profesori, maxim 3 asistenti pe curs (1 X 3, 1 X 2, 2 X 1) si 4 studenti cu 4 grupe, cate unul pe o grupa, fiecare student avand ambii parinti. In clasa test am mai adaugat pentru testare inca o grupa, inca un asistent si inca un student.

- ## Mod implementare:
  - **In mare: (1)**
  - **Clasa Catalog (1.1, 1.6)**
    - ⇨ **SINGLETON (1.1)**
      - Am facut asa cum s-a cerut in cerinta, in plus, ar fi ilogic sa existe mai multre cataloage pe an/pe serie/pe facultate etc
      - clasa contine o instanta privata Catalog care va fi creata la primul apel al functiei care returneaza catalogul, orice alta instantiere/incercare de instantiere a unui nou obiect de tipul Catalog va fi imposibila.
      - am facut totusi ca getInstace-ul sa aiba parametri, pentru a personaliza catalogul
    - ⇨ **OBSERVER: (1.6)**
      - catalogul retine o lista cu parinti care primesc notificari odata ce catalogul este actualizat;
      - cand se pune o nota si, deci, ar trebui sa apara o notificare, se parcurge lista parinti pentru a se gasi cel al carui copil a primit nota, ca sa i se trimita notificarea

- fiecare parinte in parte retine un catalog (un fel de copie a catalogului) ca sa poata primi notificare (ca sa se aboneze, cumva, la catalog)

⇨ implementeaza SUBJECT, care se intalneste in sablonul de proiectare

- **Clasa User cu Parent, Assistant, Teacher si Student cu comparatorul SortByName (1.2)**
  - **FACTORY:** datorita faptului ca exita multe clase mai multe clase derivate din User, sablonul de proiectare FACTORY ajuta la crearea de instante diferite ale clasei mari, in functie de tipul subclasei a carei instante se doreste crearea
  - Parent: implementeaza interfata OBSERVER, caci el e cel ce observa catalogul, modificarile aduse lui, deci se poate spune ca si aici se gaseste un sablon de proiectare, si anume **OBSERVER**
  - Student, Assistant implementeaza Comparable ca sa pot folosi o multime ordonata aka TreeSet, ca sa pot ordona studentii/asistentii in ordine crescatoare dupa nume (in multimea de asistenti de la acelasi curs e imposbil sa fie doi assistenti fix la fel)
  - Teacher, Assistant: implementeaza **ELEMENT,** pentru sablonul de proiectare **VISITOR,** caci el viziteaza catalogul si il modifica

- **Clasa Course cu PartialCourse si FullCourse (1.5, 1.8)**
  - implementeaza comparable tot pentru o multime ordonata cu elemente unice (am folosit multe TreeSet-uri de-a lungul temei)
  - **BUILDER (1.5)**
    - datorita faptului ca trebuie initializate multe date de clasa din Course, clasele care o implementeaza, adica PartialCourse si FullCourse, au in interiorul lor
    - clasa cu rolul de a usura procesul de instantiere, prin metode care populeaza rand pe rand campurile instantei nou create
  - **MEMENTO (1.8)**
    - pentru a se retine un snapshot din trecut al celor mai recente note dintr-un curs, pentru a preveni modificari nedorite, in interiorul clasei maeri am pus una care detine copii ale notelor dintr-un anumit moment de timp/mai multe intervale de timp, avand posibilitatea de "luare" a notelor curente din catalog "anterior", cu ajutorul unei stive pentru a lua cele mai recente date din trecut (ajuta la conservarea datelor).

- **Clasa Group (1.4)**
  - Conform cerintei, nu am nimic de adaugat in plus

- **Clasa Grade (1.9)**
  - e aproximativ ca in cerinta, numai ca am mai adaugat o noua data de clasa care ma ajuta la interfata grafica, la fila parintelui
  - in mare, Grade implementeaza Comparable si Clonable, pentru a implementa dablonul de proiectare MEMENTO si pentru a putea pune notele intr-o multime ordonata fara elemente duplicate.

Restul de design patternuri, interfete, clase:

- **Clasele BestExamScore, BestPrtialScore, BestTotalScore cu comparatorii SortingGradesE, SortingGradesP, SortingGradesT (1.7)**
  - **STRATEGY:**
    - pentru ca exista mai multe preferinte de a alege cel mai bun student, s-a folosit acest tip de sablon de proiectare, prin creearea a mai multe clase

de sine statatoare (fata de clasele mari) care contin o lista de note si depind de un curs si de un comparator.
    - De ce de un comparator? Am facut trei comparatori care ordoneaza lista de note din clasa ce implementeaza STRATEGY in functie de preferinta (pe parcurs, examen, total).
    - De obicei, notele sunt puse in ordine descrescatoare daca se ia primul element din lista, altfel crescatoare (am combinat cele doua moduri, avanmd grija sa iau maximul). (Da, poate exista functie de maxim, dar imi place se ma complit, plus ca oricum aveam nevoie de
    - comparator.)

- **Clasa Notification (1.6)**
    - Chiar daca exista deja ceva de genul in java, am facut inca o clasa Notification (am avut grija sa nu adaug o anumita biblioteca), in care am pus o nota si o functie de toString ca sa afisez nota

    - **Clasa ScoreVisitor (1.8)**
        - in metodele de vizitare mi-am adaugat date/informatii in dictionarele cu notele de la examen/pe parcurs
        - ambele au implementari asemanatoare, asa ca voi povesti numai la una: la visit de Asistant
        - am stocat intr-o variabila separata cursul din care face parte asistentul, alegandu-l dupa ce am interat prin cursulrile din catalog (a fost nevoie sa iau ca data de clasa catalogul). dintr-o a doua iterare prin cursuri am extras si grupa la care este asignat asistentul ca sa pot lua notele pe parcurs ale studentilor din acea grupa
        - tuplul l-am vazut ca doua string-uri si un grade: **<"Nume prenume", "Titlu Curs", Nota>**

- **Clasa Test (1.10)**
    - Aici am incercat sa citesc date dintr-un fisier care contine informatiile stocate astfel: pe prima linie este numarul de cursuri, numarul de cursuri PartialCourse si numarul de cursuri FullCourse, urmatoarea linie contine numele/titlul cursului, cati studenti sunt inrolati la curs si numele profesorului titular, apoi urmeaza un nr_studenti_inrolati_la_curs linii cu **Nume, Prenume, NumeTata, PrenumeTata, NumeMama, PrenumeMama, IDgrupa, NumeAsistent, PrenumeAsistent, Parcurs, Examen**
    - structura se repeta pentru fiecare curs (cea cu nume curs etc si apoi datele despre studentii inrolati)
    - am citit linie cu linie si am parsat in fucntie de spatiu, apoi am luat pentru liniile cu date despre studenti pe coloane, cum ar veni (eu le numesc coloane), si am facut cate un ArrayList de stringuri (mai putin pentru note, unde am folosit double) si le-am populat cu date, chiar daca se repetau pentru anumite cursuri.
    - apoi am avut grija sa populez catalogul, efectiv, in functie de datele din clasele Catalog si Course: pentru a tine in evidenta datele pentru fiecare curs, mi-am facut mai multe dictionare **<Curs, array_ceva_data>** in functie de ce se cere: note, studenti, asistenti, grupe etc etc

File/Ferestre/Interfata Grafica: (2)

- Daca vedeti in cele ce urmeaza ca explic ceva de o "Fereastra", ma refer, de fapt, la file. Eu le-am vazut ca niste ferestre separate, ceva interfata interesanta, cat de cat de usor de intuitiva
- Am ales ca cele trei file sa fie deschise printr-un **JDialogBox** cu un **JOptionPane (YES_NO_CANCEL)**, in care am modificat ceea ce faceau cele 3 butoane din pane.
- Nota: Fiecare fila/fereastra are un buton care face posibila intoarcerea la JDialogBox (bine, de fapt se creeaza una noua identica cu prima, mai ales ca de fiecare data cand se alege o optiune de fereastra/se apasa un buton, se inchide JDialogBox-ul) si cand se deschid mai multe ferestre (nu neaparat de acelasi tip), daca se inchide una, se inchide tot.
- Nu stiu daca e ok sa fie asa, dar nu am stiut cum sa implementez altfel.
- Fiecare fereastra/clasa de fereastra/fila are in constructor ca argumente un nume, un catalog (care in clasa Test/Main e acelasi catalog ce ramane neschimbat de la inceput (am facut asa ca sa-l dau ca argument din Test) (da, in Test, cand testez, se deschid si ferestrele, nu am vrut sa am mai multe fisiere cu main)) si un ArrayList de notificari (Cate o notificare pe student si pe curs (ca sa pun acolo tot ce se valideaza, indiferent daca e
- PartialCourse, FullCourse, vizitat de asistent sau vizitat de profesor, de la mai multe cursuri)).

- **Fila de cautare student (2.1)**
  - Aici nu am folosit vreun sablon de proiectare anume, ci pur si simplu am facut sa para ca un fel de motor de cautare de studenti care afiseaza date despre ei (mai putin parintii).
  - Am extras mai intai cursurile la care este inrolat si in fucntie de aceasta am dat ca utilizatorul sa aleaga despre ce curs al studentului sa afle mai multe folosind un ComboBox (in el am pus cursurile la care este studentul inrolat)**.**
  - Dupa ce se alege un curs din ComboBox se "definitiveaza" alegerea prin apasarea unui buton "Mai mult" care face sa se afiseze in spatiul alb, liber, datele cautate: Profesor titular, asistenti/curs, asistentul grupei studentului si notele sale. Am parsat/extras numele unui student cautat din Input/JTextField si am facut ca la apasarea butonului "Cautare" sa se itereze prin cursurile catalogului dat ca argument si daca in cursuri, printre notele studentilor cursului se numara si studentul cautat, atunci acestra se appendu-uiesc la JComboBox. Aproximativ la fel se intampla cand pentru cursul selectat extras din JComboBox si pentru studentul cautat, sa caut prin note, prin lista de asistenti si prin grupele fiecarui curs, extragand ceea ce am nevoie.
  - Daca numele studentului cautat nu este scris corect <Nume Prenume> sau nu se afla in Catalog, nu se afiseaza nicio informatie, iar ComboBox-ul nu va contine nimic (trebuie macar inrolat la un curs).

  - **Fila de Validare a notelor (2.2)**
    - Abia aici se vede adevarata functionalitate a sablonului de proiectare Visitor.
    - La selectarea butonului de validare note din JDialogBox-ul de la inceput, suntem intampinati de o fereastra (tot JDialogBox, de data aceasta cu un pane de showInputDialog) in care trebuie trecut (In mod corect, altfel nu se afiseaza nici cursurile la care este inrolat, nici, alte informatii in casetele/elementele disponibile in fereastra).
    - In functie de input vom sti daca e profesor sau asistent sau intrus (daca e intrus, intra in cazul in care nu a fost numele scris corect, deci nu a fost gasit, deci nu se afiseaza decat componetele goale ale ferestrei)

- Design-ul este asemanator celui de la Fereastra1, dar nu mai e cu bara de cautare, ci cu un combo-box cu materiile la care preda, un buton de "Mai mult" care face sa se afiseze notele de la examen/pe parcurs de la studentii la care preda (profesorul e la toti studentii din curs, asistenul numai la ce din grupa la care este asignat) si o zona in care se afiseaza datele spre a fi procesate.
- La butonul de "validare" se apleaza in spate functia de accept, care se foloseste de cea de visit din sablonul de proiectare VISITOR, acest lucru vazandu-se in linia de comanda, aratand aproape aceleasi lucruri ca la ce s-a afisat in consola la rulare, la testare. In functie de faptul ca merge sau nu, acolo unde aparut studentii si notele apare "In curs de validare..." sau "Validare realizata cu succes!" pentru a putea face validarea ok, am adaugat la toti parintii cate un catalog, le-am dat sa fie notificabili, la inceput de tot, apoi am verificat daca inputul cu numele apartine unui profesor sau al unui asistent si, in functie de aceasta am apelat functia de accept si am creat o notificare noua pe care am adaugat-o la lista globala de notificari. in acelasi timp, la creare, am avut grija sa setez data de clasa a notei notificarii in functie de tipul de persoana care vizitat (1 = profesor, 2 = asistent, 0 = default, nevizitat). Am facut cate o notificare pentru fiecare student de la acel curs/asistent in care am pus nota lui de la acael curs. a fost nevoie sa selectez din catalogul dat ca argument, acel curs selectat in comboBox.

- **Fila de notificari a parintilor (2.3)**
    - Destul de basic design-ul (recunosc, poate ca totul a fost facut pe graba, dar nici nu cred ca as fi stiut ce alt design sa mai fac). La fel ca la Validare note, se face un fel de autentificare (nu se poate numi autentificare daca nu are si o parola, dar in cerinta principala nu se cerea aceasta, ci la bonusuri) de parinte, iar imediat ce se face autentificarea, se afiseaza notificarile in functie de ce s-a validat in fila de validare note (Am vrut sa fie interactiv, cumva), aparand in chenarul alb dedicat numele cursului de la care vine notificarea, mentiunea daca e nota pe parcurs sau la examen si nota respectiva.
    - Ii dau parintelui gasit un catalog si il abonez la catlog, il fac notificabil (chiar daca probabil am mai facut asta o data)
    - Aici e putin mai usor decat la fila de validare note, cu exceptia faptului ca am luat toate cursurile si am cautat in lista de notificari data ca argument constructorului acestei ferestre daca studentul notei notificarii are ca parinte pe acela si daca da, afisez, in functie de valoarea lui ok din nota notificarea (ok = 1 examen, ok = 2 parcurs).