Tema POO

Aplicatie Catalog

Nume: Serban Andrei-Teodor

Grupa: 324CC

Grad de dificultate al temei: mediu

Timp alocat rezolvarii: 10 zile, cam 30 de ore

Rezolvare:

Clasa Catalog:

Clasa Catalog este elementul principal al aplicatiei. Aici sunt retinute toate cursurile relevante aplicatiei, dar functionalitatile incluse in clasa in sine sunt limitate. Cursurile sunt retinute intr-o colectie, courses, de tip ArrayList, iar, pe langa cursuri, catalogul mai contine un camp al clasei tine de sablonul de proiectare Observer, si retine o colectie, observers, de tip TreeSet de observatori ai catalogului, mai exact parintii studentilor inscrisi la cursurile din catalog. Ultimul camp al clasei este o instanta de tip Catalog, instance, folosita in sablonul Singleton, initializat cu null din motive ce vor fi explicate la metoda getInstance.

Clasa implementeaza interfata Subject deoarece, in cadrul sablonului de proiectare Observer, Catalogul are rolul clasei subiect, care este observat de mai multi observatori, care, la randul lor sunt notificati in momentul in care se intampla ceva in subiect (se adauga o nota noua in catalog).

Constructorul clasei instantiaza cele doua colectii agregate, iar pe langa acesta am mai implementat doua gettere, getCourses si getObservers, pentru cele doua campuri corespunzatoare.

-metoda getInstance / proprietatea Singleton:

Aceasta metoda asigura implementarea design pattern-ului Singleton pentru aceasta clasa, adica asigurarea existentei a maxim o instanta de catalog. Metoda instantiaza campul instance folosind constructorul **privat** doar daca acesta este null (metoda este apelata pentru prima data), altfel returneaza o referinta catre campul instance, astfel doar acea instanta va putea exista pe intreaga durata de rulare a aplicatiei. Metoda este statica pentru a putea fi apelata de catre un obiect, iar campul instance este static deoarece trebuie modificata in metoda statica getInstance.

-metoda addCourse:

Metoda verifica daca cursul dat ca parametru exista deja in colectia courses, iar daca nu, o adauga in aceasta.

-metoda removeCourse:

Metoda elimina un curs din colectia courses.

-metoda addObserver:

Metoda adauga un nou parinte la colectia observers.

-metoda removeObserver:

Metoda elimina un parinte din colectia observers.

-metoda notfyObservers:

Se itereaza prin toti parintii din observers, iar, pentru fiecare, apeleaza metoda update, pentru a actualiza toti observatorii subiectului cu o noua notificare, legata de adaugarea unei noi note in catalog.

Clasa User:

Clasa User este facuta sa modeleze diferitii utilizatori care pot interactiona cu catalogul in moduri foarte distincte. In implementarea clasei in sine sunt trecute cateva trasaturi comune de baza, exclusiv legate de numele utilizatorilor. Acesta contine doua campuri de tip String, firstName si lastName, pentru prenumele si numele fiecarui utilizator.

Clasa implementeaza interfata Comparable pentru a se putea asigura o diferentiere intre utilizatori in colectiile in care acestia sunt agregati, bazata pe nume si prenume.

Constructorul cu doi parametri seteaza cele doua campuri ale clasei, iar getLastName exista doar pentru un comparator personalizat folosit de mine la testare.

-metoda compareTo:

Aceasta este suprascrisa pentru a asigura compararea utilizatorilor pe baza de combinatia prenume + nume.

-metoda equals:

Similar, egaleaza doi utilizatori daca acestia impart combinatia de prenume + nume.

-metoda toString:

Este suprascrisa, ea returneaza combinatia dintre campurile clasei, separate de un spatiu.

Clasa Parent:

Aceasta clasa este o subclasa a clasei User si modeleaza unul din cele patru tipuri de utilizatori utilizate de aplicatie, cel al parintelui, care, in cea mai mare parte, exista pentru a-si indeplini rolul de observator in implementarea sablonului Observer. Din acest motiv clasa implementeaza si interfata Observer. In clasa exista un singur camp, o colectie, notifications, de tip TreeSet care retine toate notificarile primite de un parinte sub forma de mesaje text.

Clasa contine un constructor care, in afara de a apela constructorul clasei parinte cu aceiasi parametri, instantiaza si colectia de notificari.

-metoda update:

Aceasta metoda adauga un mesaj nou de notificare legata de actualizarea unei note, doar daca parintele din care e apelata metoda este parintele studentului a carui nota a fost actualizata.

-metoda showNotifications:

Metoda afiseaza toate elementele colectiei de mesaje notifications.

Clasa Student:

Clasa Student mosteneste clasa User si este facuta pentru a modela alt tip de utilizator al catalogului, studentul, care primeste note si e inscris la cursuri. Deosebit de celelalte tipuri de utilizatori, clasa Student are doua campuri de tip Parent: unul pentru mama studentului, altul pentru tatal studentului.

Constructorul clasei doar apeleaza constructorul clasei parinte, si mai sunt implementati doi getteri (getMother, getFather) si doi setteri (setMother, setFather) pentru cele doua campuri ale studentului.

Clasa Assistant:

Clasa Assistant extinde clasa User si modeleaza un tip de utilizator al catalogului, asistentul, care pune note partiale studentilor, este titular pe grupe de studenti dar **NU** si pe cursuri.

Clasa implementeaza interfata Element, din cadrul sablonului de proiectare Visitor, deoarece este o clasa care poate accepta vizite de la vizitator (ScoreVisitor-ul care trece notele in catalog), confirmand notele puse in catalog de sine.

Constructorul clasei doar apeleaza constructorul superclasei.

-metoda accept:

Este o metoda standard pentru sablonul Visitor, facand visit din Visitor-ul dat ca argument la obiectul din care este apelata metoda.

Clasa Teacher:

Clasa Teacher extinde clasa User si modeleaza alt tip de utilizator al catalogului, profesorul de curs, care este foarte similar cu asistentul, pune note studentilor in examen si este titular pe un intreg curs.

Clasa implementeaza interfata Element din aceleasi motive ca Assistant, singura diferenta fiind ca Teacher valideaza note in examen in timp ce Assistant valida note obtinute pe parcursul semestrului.

-metoda accept:

La fel ca la Teacher, face visit din argument in obiectul din care se apeleaza metoda.

Clasa UserFactory:

Clasa UserFactory are rolul de a asigura indeplinirea conditiilor pentru sablonul de proiectare Factory, de creatie a diferitilor utilizatori. Exista patru tipuri permise de utilizatori: “Student”, “Parent”, “Assistant” si “Teacher”, fiecare tip pentru clasa dupa care e numit.

-metoda getUser:

Este o metoda statica, care returneaza cate un obiect nou de tipul mentionat in argumentul type, folosind constructorii fiecarei subclase a lui User in parte, cu argumentele date tot ca parametri metodei. In cazul in care argument-ul type nu se regaseste in cele 4 tipuri mentionate, se arunca o exceptie de tipul IllegalArgumentException.

Clasa Grade:

Clasa Grade modeleaza nota care se pune in catalog unui student, impartita in doua: punctajul obtinut pe parcursul semestrului si nota din examen. Clasa are doua campuri Double, partialScore si examScore, care reprezinta nota pe parcurs, respectiv nota in examen, un camp de tip Student, student, care reprezinta studentul care a luat nota si un camp de tip String, course, care reprezinta numele cursului la care s-a luat nota.

Clasa implementeaza interfetele Comparable, pentru a putea exista un mod de a diferentia notele intre ele (dupa nota totala) si Cloneable, pentru a putea fi facuta o copie dupa nota ce va ajuta la sablonul de proiectare Memento.

Clasa are doi constructori, unul care ia ca argumente valori ce corespund tuturor campurilor clasei si i le seteaza, si unul care ia ca argument alt grade si ii copiaza toate campurile. Pe langa acestia, clasa mai are si getteri si setteri pentru toate campurile.

-metoda compareTo:

Metoda suprascrisa pentru a compara doua note dupa valoarea lor totala.

-metoda equals:

Metoda care returneaza adevarat doar daca valorile pentru partialScore **SI** pentru examScore sunt egale simultan intre cele doua obiecte.

-metoda getTotal:

Metoda care returneaza suma dintre partialScore si examScore, aceasta reprezentand nota finala a studentului.

-metoda toString:

Suprascrisa pentru a scrie nota intr-un format in care se evidentiaza nota pe parcurs, nota din examen si nota finala.

Clasa Group:

Clasa Group mosteneste clasa ArrayList<Student> deci functioneaza ca o extindere a unei liste de studenti. Aceasta modeleaza o grupa de studenti, cu un nume unic, un asistent titular si o

modalitate de sortare. Campurile clasei sunt: unul de tip String, ID, care reprezinta numele grupei si are rol de identificator unic, unul de tip Assistant, assistant, care reprezinta asistentul titular acelei grupe si unul de tip Comparator<Student>, comparator, care reprezinta comparatorul dupa care ar putea fi sortati studentii in acea grupa.

Clasa are doi constructori, unul care ia ca argumente valori pentru toate campurile clasei si unul care omite comparatorul, acesta fiind setat ca ceea ce eu am decis sa fie ordinea naturala a clasei, ordinea alfabetica a studentilor (dupa prenume inainte de nume, deoarece apelez metoda compareTo din Student care asa ii compara). Ambii constructori apeleaza constructorul superclasei inainte de a seta cele trei campuri. Pe langa acesti constructori mai exista getteri pentru toate campurile si un setAssistant pentru a seta asistentul grupei cu altul.

-metoda toString:

Returneaza un format in care sunt delimitate numele grupei, asistentul responsabil si lista de studenti.

Clasa CourseBuilder:

Clasa CourseBuilder este o clasa abstracta, statica si interna clasei Builder si modeleaza un builder folosit pentru sablonul de proiectare Builder. Aceasta contine toate campurile clasei Course, pe care le modifica printr-o serie de setteri modificati care returneaza tot un obiect de tip CourseBuilder, dar cu un singur camp modificat / ”setat”. Aceasta este si o clasa abstracta si contine antetul metodei abstracte build care returneaza obiectul de tip Course construit in totalitate, sau doar cu cateva campuri dorite setate. Metoda si, drept consecinta, clasa sunt abstracte deoarece Course are doua subclase, fiecare dintre ele avand un builder intern, care mosteneste CourseBuilder, asadar metoda build trebuie lasata implementata la latitudinea tipului de curs care va fi construit.

Clasa Snapshot:

Clasa Snapshot este o clasa interna privata a clasei Course care face parte din design pattern-ul Memento. Aceasta modeleaza o colectie de note de backup, pentru eventuala revenire la notele vechi dupa modificarea lor. Clasa are un singur camp, un ArrayList<Grade>, grades, care retine lista de note de backup. Singura metoda din clasa este un setter care, in loc sa copieze referinta colectiei data ca parametru, copiaza toate elementele unul cate unul cu ajutorul metodei clone din clasa Grade, pentru a nu se transfera toate modificarile de note.

Clasa Course:

Clasa Course modeleaza cursul la care sunt inscrisi studentii, care este tinut de profesor si care are mai multi asistenti. Este probabil a doua cea mai esentiala clasa pentru aplicatie, dupa Catalog. Campurile clasei sunt un String, name, pentru numele cursului, un Teacher, teacher, pentru profesorul de curs, un TreeSet<Assistant>, assistants, pentru multimea asistentilor, un ArrayList<Grade>, grades, pentru lista tuturor notelor de la acel curs, un Hashmap<String, Group>, groupMap, un dictionar in care oricarui nume de grupa ii este asociat obiectul Group cu acel nume, un int, credits, pentru numarul de credite pe care il acorda absolvirea materiei, un

Strategy, strategy, pentru strategia dupa care profesorul isi alege cel mai bun student, parte din implementarea sablonului de proiectare Strategy si un Snapshot, snapshot, initial egal cu null, pentru o instanta a listei de note de backup, in cazul in care se doreste la revenirea la notele anterioare, parte din implementarea design pattern-ului Memento.

Constructorul clasei ia ca argument un obiect de tip CourseBuilder, si este de fapt folosit doar in implementarile din subclase ale metodelor build, aceasta fiind metoda care este folosita la testare pentru a instantia obiecte de tip Course, deoarece asta este regula design pattern-ului Builder. Constructorul este de tipul protected pentru a putea fi accesat doar de subclasele sale. Mai sunt si getteri pentru toate campurile clasei, mai putin strategy si snapshot si un setter pentru strategy.

-metoda addAssistant:

Metoda seteaza in groupMap pentru grupa cu numele ID asistentul assistant si, in cazul in care nu e parte din set-ul de asistenti, il adauga. Un add fara verificari este suficient deoarece clasa TreeSet sterge oricum duplicatele automat.

-metoda addStudent:

Metoda adauga, folosind groupMap-ul, un student nou student la grupa cu numele ID.

-metodele addGroup:

Sunt trei metode supraincarcate cu numele addGroup care iau parametri diferiti. Prima dintre ele ia ca parametru un obiect Group pe care il pune direct in groupMap si, folosindu-i getterii, adauga asistentul grupei in set-ul de asistenti, pentru cazul in care acesta nu exista deja acolo. Cea de-a doua metoda ia ca argumente un String ID pentru numele grupei, un Assistant assistant pentru asistentul responsabil pentru grupa si un Comparator<Student> comparator pentru criteriul dupa care va fi sortata grupa, in cazul in care ea va fi sortata. Aceasta metoda face aceiasi pasi ca si prima, doar ca apeleaza constructorul cu comparator al clasei Group pentru a si instantia o grupa noua. Cea de-a treia metoda ia ca argumente doar un String ID pentru numele grupei si un Assistant assistant pentru asistentul responsabil si e identica cu cea de-a doua metoda, doar ca apeleaza constructorul fara parametru al clasei Grade.

-metoda getGrade:

Metoda cauta in toata lista de note grades si returneaza nota a carui getter corespunde studentului dat ca argument. In cazul in care nu gaseste nota pentru studentul respectiv, returneaza null.

-metoda addGrade:

Adauga o nota noua in colectia de note.

-getAllStudents:

Itereaza prin valorile dictionarului groupMap, iar apoi prin fiecare grupa in parte si adauga toti studentii gasiti intr-o colectie de tip ArrayList<Student> pe care o returneaza.

-getAllStudentGrades:

Itereaza prin lista de note si creeaza un HashMap<Student, Grade> pentru toti studentii si notele pe care le au, pe care apoi il returneaza.

-getGraduatedStudents:

Metoda abstracta a carei implementari se afla in clasele FullCourse si PartialCourse.

-getBestStudent:

Metoda legata de sablonul de proiectare Strategy ce returneaza rezultatul metodei getBestGrade din campul strategy al clasei, rezultat ce poate varia in functie de strategia aleasa de profesorul de curs.

-makeBackup:

Metoda legata de sablonul de proiectare Memento ce instantiaza campul snapshot pentru a indica metodei undo ca s-a facut un backup si seteaza colectia de note din snapshot cu notele curente din curs.

-undo:

Metoda legata de sablonul de proiectare Memento ce verifica intai daca exista un backup facut (daca snapshot nu e null) si apoi reseteaza notele din campul grades cu cele din snapshot.

Clasa PartialCourseBuilder:

Clasa PartialCourseBuilder este clasa interna a clasei PartialCourse, care mosteneste clasa CourseBuilder si ii implementeaza metoda abstracta build, returnand un nou curs PartialCourse. Clasa modeleaza un builder personalizat pentru obiecte de tipul PartialCourse.

Clasa PartialCourse:

Clasa PartialCourse mosteneste clasa Course si ii implementeaza metoda abstracta getGraduatedStudents. Clasa modeleaza cursul partial, unde conditia de trecere este ca nota totala sa fie peste 5.

Constructorul este privat pentru a putea fi accesat doar de clasa sa interna PartialCourseBuilder.

-metoda getGraduatedStudents:

Adauga intr-un ArrayList<Student> toti studentii care au nota totala peste 5, aceasta reprezentand lista studentilor promovati la un curs partial, care este returnata de metoda.

Clasa FullCourseBuilder:

Clasa FullCourseBuilder este clasa interna a clasei FullCourse, care mosteneste clasa CourseBuilder si ii implementeaza metoda abstracta build, returnand un nou curs FullCourse. Clasa modeleaza un builder personalizat pentru obiecte de tipul FullCourse.

Clasa FullCourse:

Clasa FullCourse mosteneste clasa Course si ii implementeaza metoda abstracta getGraduatedStudents. Clasa modeleaza cursul intreg, unde conditia de trecere este ca simultan nota pe parcurs sa fie mai mare de 3 si nota din examen sa fie mai mare de 2.

Constructorul este privat pentru a putea fi accesat doar de clasa sa interna FullCourseBuilder.

-metoda getGraduatedStudents:

Adauga intr-un ArrayList<Student> toti studentii care au nota pe parcurs peste 3 si nota din examen peste 2, aceasta reprezentand lista studentilor promovati la un curs intreg, care este returnata de metoda.

Interfata Observer:

Expune antetul metodei update, implementata de clasa Parent.

Interfata Subject:

Expune antetele metodelor addObserver, removeObserver si notifyObservers, implementate de clasa Catalog.

Clasa Notification:

Clasa Notification modeleaza notificarea pe care o primesc observatorii (parintii) subiectului (catalogul). Clasa contine un camp privat de tip Grade, grade, care retine nota care ar trebui anuntata parintilor.

Constructorul clasei este unul care seteaza nota cu cea data ca parametru. In plus, mai e implementat un getter pentru aceasta nota.

-metoda toString:

Returneaza un mesaj de notificare in care este adresat carui student i-a fost actualizata care nota la care curs.

Interfata Strategy:

Expune antetul metodei getBestGrade, implementata de clasele BestExamScore, BestPartialScore si BestTotalScore.

Clasa BestExamScore:

Clasa BestExamScore modeleaza strategia profesorului conform careia cel mai bun student la materia sa este cel cu nota maxima in examen. Clasa implementeaza interfata Strategy si reprezinta una din cele 3 strategii din implementarea ceruta a sablonului de proiectare Strategy.

-metoda getBestGrade:

Itereaza prin notele studentilor inscrisi la un curs si retine un maxim local pe care il actualizeaza la fiecare nota de examen mai mare pe care o gaseste. La final returneaza studentul pentru care acea nota a fost maxima.

Clasa BestPartialScore:

Clasa BestPartialScore modeleaza strategia profesorului conform careia cel mai bun student la materia sa este cel cu nota maxima pe parcurs. Clasa implementeaza interfata Strategy si reprezinta una din cele 3 strategii din implementarea ceruta a sablonului de proiectare Strategy.

-metoda getBestGrade:

Itereaza prin notele studentilor inscrisi la un curs si retine un maxim local pe care il actualizeaza la fiecare nota pe parcurs mai mare pe care o gaseste. La final returneaza studentul pentru care acea nota a fost maxima.

Clasa BestTotalScore:

Clasa BestTotalScore modeleaza strategia profesorului conform careia cel mai bun student la materia sa este cel cu nota maxima per total. Clasa implementeaza interfata Strategy si reprezinta una din cele 3 strategii din implementarea ceruta a sablonului de proiectare Strategy.

-metoda getBestGrade:

Itereaza prin notele studentilor inscrisi la un curs si retine un maxim local pe care il actualizeaza la fiecare nota finala mai mare pe care o gaseste. La final returneaza studentul pentru care acea nota a fost maxima.

Interfata Element:

Expune antetul metodei accept, implementata de clasele Teacher si Assistant.

Interfata Visitor:

Expune antetele metodelor visit supraincarcate (una pentru Teacher, una pentru Assistant), implementate de clasa ScoreVisitor.

Clasa Tuple:

Clasa Tuple este o clasa interna privata si generica a clasei ScoreVisitor care modeleaza un triptic de obiecte de trei tipuri diferite. Clasa are un constructor cu trei argumente care seteaza cele trei campuri ale ei, precum si 3 getteri pentru fiecare camp in parte. In implementarea din ScoreVisitor va fi nevoie de Tuple-uri de <Student, String, Double> care retin studentul care a primit nota de tip double la cursul cu numele de tip String.

Clasa ScoreVisitor:

Clasa ScoreVisitor modeleaza entitatea care transfera notele din doua dictionare locale in catalog, folosindu-se de sablonul de proiectare Visitor. Cele doua dictionare sunt un Hashmap<Teacher, ArrayList<Tuple<Student, String, Double>>> examScores, care retine toate notele la examen din Tuple care au fost puse de profesorul din cheia dictionarului studentului aflat in Tuple la materia cu numele aflat tot in Tuple, si Hashmap<Assistant, ArrayList<Tuple<Student, String, Double>>> partialScores care retine un lucru similar, doar ca

referitor la notele pe parcurs puse de fiecare asistent. Aceste doua dictionare sunt umplute la citirea datelor din fisierele de test si apoi datele sunt transferate in catalog prin intermediul metodelor de visit din interfata pe care o implementeaza.

Constructorul clasei instantiaza cele doua dictionare. In plus, exista un getter pentru fiecare dictionar.

-metoda addNewExamScoreEntry:

Are 4 parametri: un profesor de tip Teacher, un student de tip Student, un nume de curs de tip String si o nota de tip Double. Scopul metodei este sa adauge in dictionarul de note de examen, la cheia profesorului dat ca parametru, un nou Tuple ce concateneaza celelalte 3 argumente. Acest pas nu se poate face in metoda de testare deoarece clasa Tuple este impusa ca fiind privata, asadar nu se poate nicaieri altundeva decat in interiorul clasei ScoreVisitor sa se instantieze un nou Tuple. Asadar, metoda itereaza prin hashmap si, cand gaseste profesorul egal cu cheia din hashmap, se duce la valoare si adauga un nou Tuple cu argumentele mentionate anterior.

-metoda addNewPartialScoreEntry:

Are 4 parametri: un asistent de tip Assistant, un student de tip Student, un nume de curs de tip String si o nota de tip Double. Scopul metodei este sa adauge in dictionarul de note partiale, la cheia asistentului dat ca parametru, un nou Tuple ce concateneaza celelalte 3 argumente. Acest pas nu se poate face in metoda de testare deoarece clasa Tuple este impusa ca fiind privata, asadar nu se poate nicaieri altundeva decat in interiorul clasei ScoreVisitor sa se instantieze un nou Tuple. Asadar, metoda itereaza prin hashmap si, cand gaseste asistentul egal cu cheia din hashmap, se duce la valoare si adauga un nou Tuple cu argumentele mentionate anterior.

-metoda visit(Teacher):

Metoda transfera notele din examScores in catalogul gol, creand note noi de tip Grade doar atunci cand este nevoie. In primul rand metoda retine intr-o variabila tempCourses lista de cursuri din catalog. Se itereaza prin Hashmap pana cand se ajunge la cheia egala cu argumentul. De acolo se itereaza prin lista de note care trebuie sa fie puse in catalog de profesor, sub forma de Tuple. Se creeaza mai multe variabile auxiliare si apoi se itereaza prin toate cursurile. Inainte de a gasi numele cursului dat in Tuple in toata lista de cursuri, se mai face o iterare prin dictionarul groupMap al fiecarui curs in parte pentru a gasi studentul care este exact cel din Tuple si a-i schimba direct referinta de obiect cu cea din groupMap. Acest pas este facut deoarece, la partea de testare, referintele la studentii carora le-au fost setati corespunzator parintii sunt cei din groupMap-uri, iar cum este nevoie in aceasta metoda de parinti pentru partea legata de sablonul Observer, cel mai usor mod de a transfera parintii a fost acela de a modifica asa referinta direct. Dupa ce se gaseste cursul din Tuple-ul curent se verifica daca metoda getGrade(Student) din curs returneaza null. In cazul in care returneaza null, inseamna ca inca nu s-a facut un obiect Grade care sa retina nota de examen la materia respectiva, asa ca se adauga o instanta noua de Grade. In caz contrar, doar se modifica nota la examen, folosind getGrade si setter-ul din clasa Grade. Ultimul lucru inainte sa treaca la urmatoarea nota ce trebuie adaugata

este sa notifice toti observatorii ca a fost adaugata o noua nota. Aceasta notificare noua este filtrata la nivelul metodei update al fiecarui observator in parte.

-metoda visit(Assistant):

Metoda transfera notele din partialScores in catalogul gol, creand note noi de tip Grade doar atunci cand este nevoie. In primul rand metoda retine intr-o variabila tempCourses lista de cursuri din catalog. Se itereaza prin Hashmap pana cand se ajunge la cheia egala cu argumentul. De acolo se itereaza prin lista de note care trebuie sa fie puse in catalog de asistent, sub forma de Tuple. Se creeaza mai multe variabile auxiliare si apoi se itereaza prin toate cursurile. Inainte de a gasi numele cursului dat in Tuple in toata lista de cursuri, se mai face o iterare prin dictionarul groupMap al fiecarui curs in parte pentru a gasi studentul care este exact cel din Tuple si a-i schimba direct referinta de obiect cu cea din groupMap. Acest pas este facut deoarece, la partea de testare, referintele la studentii carora le-au fost setati corespunzator parintii sunt cei din groupMap-uri, iar cum este nevoie in aceasta metoda de parinti pentru partea legata de sablonul Observer, cel mai usor mod de a transfera parintii a fost acela de a modifica asa referinta direct. Dupa ce se gaseste cursul din Tuple-ul curent se verifica daca metoda getGrade(Student) din curs returneaza null. In cazul in care returneaza null, inseamna ca inca nu s-a facut un obiect Grade care sa retina nota pe parcurs la materia respectiva, asa ca se adauga o instanta noua de Grade. In caz contrar, doar se modifica nota pe parcurs, folosind getGrade si setter-ul din clasa Grade. Ultimul lucru inainte sa treaca la urmatoarea nota ce trebuie adaugata este sa notifice toti observatorii ca a fost adaugata o noua nota. Aceasta notificare noua este filtrata la nivelul metodei update al fiecarui observator in parte.

Clasa Test:

Clasa Test este clasa folosita pentru a crea entry point-ul aplicatiei, o metoda main.

-metoda main:

Am folosit ca fisier de intrare un fisier de tip .json, pe care l-am parsat cu API-ul json-simple. Asadar prima parte din main este parsarea fisierului folosind foarte multe JSONObject-uri si JSONArray-uri. Partea de introducere de cursuri a fost relativ usoara deoarece pur si simplu se copia din fisierul de intrare si se adauga in Catalog cu toti setterii builder-ului. La partea cu notele a intervenit mai multa logica. Fiecare profesor punea mai multe note, asadar am verificat cu ajutorul unei variabile hashFlag daca profesorul exista deja in keySet-ul hashMap-ului. In cazul in care nu era, adaugam o noua intrare in dictionar, cu valoarea un ArrayList gol. Acelasi lucru l-am facut si pentru notele asistentilor. Apoi, dupa fiecare set de note introduse in catalog am acceptat vizitatorul pentru fiecare profesor / asistent in parte, astfel transferand notele in catalog. Ultimul lucru pe care il face metoda main este sa testeze toate metodele descrise in acest document, aproximativ in ordinea in care le-am descris.

Concluzie:

A fost o tema foarte interesanta care m-a facut sa inteleg cum intervin conceptele paradigmei orientata pe obiecte intr-un proiect mai stufos. Singura problema pe care am avut-o a fost enuntul vag la cerintele cu design patterns. Inteleg ideea de a lasa ceva mai ambiguu pentru a spori creativitatea fiecarui student de implementare, dar din ce am vorbit si cu colegii mei s-a creat mai mult o ceata de neclaritate decat un challenge distractiv ce impune o gandire out of the box. Cred ca absolut nimeni n-a inteles rostul design pattern-ului Visitor si de ce notele nu puteau fi direct transferate in catalog, fara intermediarul celor doua dictionare din ScoreVisitor.

Regret totusi ca nu m-am apucat de tema mai din timp, sa fi avut timp sa fac si interfata grafica si sa inteleg conceptele de Swing pe care oricum nu le stapanesc prea bine.