README

Chitu Andrei Alexandru 321 - CC
Dificultate : medie
Dificultate intelegere cerinta : grea (spre foarte grea)

Clase :

Catalog :
Clasa catalog este clasa in care am retinut array-ul de cursuri. Aceasta a avut o implementare speciala deoarece a m folosit sablonul de proiectare
Singleton pentru a putea creea o singura instanta de catalog. Astfel am definit un catalog ca fiin null la inceput si a m creeat o metoda numita
getInstance care returneaza un nou catalog in cazul in care acesta nu este null. Prin urmare vom putea instanta Cat alog o singura data, atunci cand
acesta este null. Suplimentar fata de metodele cerute am adaugat o metoda getCourse care primeste un string num e_curs ca parametru si returneaza
obiectul de tip curs cu acel nume sau null
AddCourse adauga la arrayul de cursuri cu .add
RemoveCourse scoate din arrayul de cursuri cu .remove

User,Student,Assistent,Parent,Teacher :
Clasele student, asistent si parinte sunt clase ce mostenesc clasa user. Pentru aceasta clasa am avut de implementat o clasa speciala numita UserFactory ce
are o metoda getUser. Practic aceasta metoda primeste ca parametru "Student, nume, prenume" si in interiorul ei c ompara primul camp(type), care in exemplul
de mai sus este student sa vada ce obiect returneaza. Poate returna 4 typeuri Student,Asistent,Parinte si Profesor. Astfel instantierea unui obiect
este mai usoara deoarece orice obiect de tip user fie el student asistent etc se instantiaza cu ajutorul userFactory.
SetFather = this.father = father la fel si pentru mother

Grade :
Clasa grade implementeaza practic informatiile despre un student si nota pe care a primit-o. Astfel am facut getteri si setteri pentru examScore si Partial
Score si pentru getTotal le am adunat si returnat. Am creeat un comparator care sa compare dupa .getTotal() si am creat o functie ce cloneaza un obiect
de tip grade. Practic retin din nou toate campurile intr-un alt obiect de tip Grade coppy si returnez coppy;

Group :
Clasa Group mosteneste o colectie ordonata cu obiecte de tip Student. Am facut un comparator ce sorteaza student ii din grupe in ordine lexico-grafica
si am setat campurile ca in cerinta

Course :
Clasa Course implementeaza practic toate clasele de mai sus.
Pentru addAssistent am folosit un Set care nu retine dublicate pentru a adauga si am adaugat in Set<Assistent> cu .add
Pentru addStudent am iterat prin grupe, am comparat ID ul primit ca parametru cu ID ul grupei si daca erau la fel am adaugat cu addStudent din Group
Functiile de addGroup adauga cu put la HasMapul groups
Functia getGrade parcurge vectorul de grades si returneaza daca studentul primit parametru e egal cu vreunul din s et
Functia addGrade adauga cu add la vectorul de grade
Functia getAllStudents itereaza prin grades si adauga la un array local studentul folosind functia getStudent din gr ade apoi returneaza arrayul

Functia gettAllStudents face acelasi lucru dar creeaza un hasmap unde pune grade.getStudent ca cheie si grade ca valoare

Builder :

Builderul este un sablon util de folosit pentru instantiere. Astfel am creeat cum mi s-a cerut clasele interne dupa ca re precum la un constructor

cu ajutoru this am atribuit valorile. Practic cel din Course returneaza this, iar builderii din celelalte clase il foloses c pt instantiere

FullCourse, PartialCourse :

Clase inutile care nu ajuta cu nimic, au practic aceleasi campuri doar ca au fost date banuiesc pentru a folosi build erul

Diferenta ar fi totusi la getGraduated students pentru ca se tine cont si de partial la FullCourse

Observer :

Sablonul observer este folosit pentru a trimite notificari. Astfel daca un elev primeste o nota o notificare este trimi sa. Nu ar fi logic sa verificam

incontinuu daca elevul in cauza a primit sau nu o nota. Clasa Parent implementeaza Observer ul pentru ca parintel e este un observator

Am creeat un vector de notificari in parinte si o functie de update care stocheaza notificarile primite de un parinte. Practic cand folosesc update

se adauga o notificare parintelui.

Subject este implementata de catalog deoarece in catalog se adauga observatorii (parintii)

Astfel avand observatorii in catalog, cand adaugam o nota putem trimite o notificare la parintele studentului daca este un observator despre nota

pe care acesta a luat-o

Strategy :

Sablonul Strategy este folosit pentru a nu fi nevoit sa lucrezi cu 3 functii de o data. Astfel avem definite clase pent ru fiecare strategie si

interfata strategy. Aceasta este construita in constructorul builder. Spre exemplu daca strategia este X atunci se ret urneaza un obiect de tip

X si strategia cand este apelata X.faCeva(), va apela functia faCeva din clasa X. Daca strategia era Y atunci apelul strategy.faCeva apela

functia faCeva din Y;

Vistior :

Visitor este un sablon care ajuta validarea, sau mai bine zis vizitarea / modificarea notelor. Element in cazul nostr u este elementul care viziteaza

Astfel clasa parinte si clasa assistent implementeaza element. ScoreVisitor are 2 mapuri care sunt setate cu 2 meto de creeate de mine setPartial, setExam

Practic acolo retinem notele pe care vrea sa le puna un asistent sau un profesor. Dupa care implementarea efectiva este ca atunci cand

gasim o nota ce nu a fost pusa unui elev o adaugam sau daca o nota este modificata o modificam cu setPartial/Exa mScore.

Memento:

Cu sablonul memento creez o copie a notelor astfel am functia de makeBackup care parcurge grade urile din grade s si le adauga la un snapshot care

este de fapt o clasa ce contine un treeset de grade-uri, iar pentru undo grade-urile din curs = snapshot.grades