Exercitii - Programare Avansata pe Obiecte

Nicolae Marius Ghergu

March 29, 2023

1. Scrieti un program care sa afișeze toate numerele pare din intervalul [0,n], unde n este un numar citit de la tastatura.

2. Scrieți un program care sa compare doua numere a și b citite de la tastatura si sa afiseze numarul mai mare.

3. Scrieți o metoda care sa calculeze factorialul unui numar n citit de la tastatura.

4. Fiind dat un numar n, scrieti o metoda care insumeaza toti multiplii de 3 si 5 pana la n (inclusiv).

5. Cititi de la tastatura n numere. Elementele citite vor fi organizate in doi vectori diferiti, in functie de paritate (de ex: elementele pare in vectorul x, iar cele impare in vectorul y).

6. Cititi de la tastatura n note, numere intregi, intr-un vector. Cand cititi valoarea -1 de la tastatura, citirea notelor se opreste si programul afiseaza media acestora.

7. Creeaza un program care contine o clasa 'Person' cu urmatoarele atribute: name as string, surname as string, age as int, identity number as long, type as string. Defineste un constructor pentru clasa, getteri si setteri. Creeaza doua obiecte ale tipului 'Personala' si afiseaza informatia in linii separate. Creeaza o clasa 'Room' cu urmatoarele atribute: room number, room type and room floor. Defineste un constructor pentru aceasta clasa, getteri si setteri pentru toate atributele. Creeaza doua obiecte de tipul 'Room si afiseaza informatia pe linii separate. Creeaza o clasa 'Subject' cu urmatoarele atribute: room as Room, noOfStudents as integer, teacher as Person. Creeaza un constructor, getteri si setteri pentru toate atributele. Creeaza doua obiecte de tip Subject si afiseaza informatia pe linii separate. Creeaza o clasa Meniu care sa fie singleton in care introduci toate operatiile facute pe clasele definite si o metoda principala care este apelata din clasa Main.

8. Proiectati o clasa CandyBox, care va continue campurile: flavor (String), origin (String) cu modificatorii de access corespunzatori. Clasa va avea de asemenea:
a. Constructor fara parametri
b. Constructor care va initializa toate campurile
c. O metoda abstracta getVolume()
d. Suprascrierea metodei toString()
Din ea derivati clasele Merci, Lindt, Milka. Pentru un design diferit, cutiile au diverse forme:
a. Lindt va continue length, width si height
b. Milka va fi un cilindru cu campurile radius si height
c. Merci va fi un cub si va contine campul length
Clasele vor avea de asemenea:
a. Constructor fara parametri
b. Constructor care initializeaza membrii claselor
c. Suprascrierea metodei getVolume()
d. Suprascrierea metodei toString() care sa returneze un mesaj de genul: "The " + origin + " " + flavor + " has volume " + volume.
Verificati egalitatea acestor obiecte create (din fiecare tip) si adaugati metoda equals() dupa cum este nevoie. Justificati criteriul de echivalenta ales.

Creati o clasa CandyBag care va contine un array cu cutii din fiecare tip.

Creati clasa Area care va continue un obiect de tip CandyBag, un camp number (int) si unul street (String). Clasa va contine:
a. Constructor fara parametri
b. Constructor care initializeaza atributele
c. O metoda printAddress() care va afisa adresa completa si continutul CandyBag (parcurgeti array-ul si apelati metoda toString() pentru elementele sale).

9. Se citeste un sir de caractere de la tastatura, verificati daca este un palindrom.

10. Scrieti un program care verifica daca doua siruri de caractere sunt anagrame. (ex: ramo, mora si roma sunt anagrame).

11. Sa se implementeze o clasa PasswordMaker ce genereaza o parola pornind de la datele unei persoane. Aceasta clasa o sa contina urmatoarele:
a. o constanta MAGIC_NUMBER, care ia orice valoare doriti
b. un String constant MAGIC_NUMBER, lung de minim 20 caractere, generat random (puteti crea o metoda care da un caracter random)
c. un constructor care primeste un String, numit name
d. o metoda getPassword() care va returna parola
Parola se construieste concatenand urmatoarele:
- un sir random de lungime MAGIC_NUMBER
- 10 caractere din MAGIC_NUMBER
- lungimea atributului name ca si String
- un numar intreg generat random din intervalul [0,100] folosind clasa Random
Modificati clasa PasswordMaker astfel incat sa respecte conceptul de Singleton.

12. Declarati o interfata Task care contine o metoda execute(), care returneaza void. Pe baza acestei interfete implementati 3 clase: RandomTask, OutTask si CounterOutTask.
a. Pentru OutTask afisati un mesaj in consola, mesaj specificat n constructor
b. Pentru RandomTask generati un numar aleator si afisati un mesaj cu el. Generarea se face in constructor
c. Pentru CounterOutTask, incrementati un contor global si afisati-i valoarea dupa fiecare incrementare
Creati o noua clasa Container in care puteti adauga si elimina elemente.

13. Declarati o clasa Album care are campurile: nume, artist, rating si anul publicarii.
a. Sortati un array de albume pe baza numelui, rating-ului si anului publicarii. Folositi ambele interfete de comparare.
b. Creati o clasa Main unde declarati array-ul si afisati-l inainte si dupa sortare.

14. Creati 4 interfete Minus, Plus, Mult si Div care contin cate o metoda aferenta numelui si are ca argument un numar de tipul float. Declarati o clasa Operation care sa le implementeze si care are un camp de tip float, modificat de metodele implementate de voi.

15. Creeaza o clasa 'Book' cu campurile: titlu, autor si anul publicatiei.
a) In Menu/Main class adauga 7 carti in lista respectiva.
b) Alege toate cartile pentru a avea numai pe cele care au anul publicatiei inainte de 2000.
c) Printeaza anul publicarii celor ramase.
d) Fa asta folosind stream-uri: stream, filter, sorted and forEach methods.

16. Creeaza o clasa 'Calendar' prin care sa stochezi ziua, luna si anul in care ne aflam. Clasa trebuie sa fie una singleton. Creeaza o solutie pentru a putea stoca si timpul. Gaseste o solutie pentru a face calendarul sa fie unul funtional. (Sa incrementeze automat cate 1 secunda pe ceas). Tip: foloseste 'while true'.

16-a. Creeaza problema anterioara folosind clasele LocalDate si LocalDateTime.

16-b. Creeaza problema 16 folosindu-te si de clasa Instant. (Instant.now() -¿ pentru a lua momen-

tul actual. Exista si metoda System.currentTimeMills()).

17. Implementeaza o solutie folosind structura layerizata (model-service, menu singleton class), pentru a implementa un mini-sistem bancar. O sa ai o clasa 'Account' prin care vrem sa stocam informatiile conturilor clientilor nostri, clasa 'Client' prin care ne definim toti clientii existenti. Exista relatie one-to-many.

Clasa AbstractEntity este o clasa abstracta care contine id-ul, creationDate, disableDate si updatedDate.

Clasa Account contine id-ul contului, suma contului, alias-ul, id-ul clientului, tipul de cont si numarul contului (16 cifre).

Clasa Client contine informatii despre client cum ar fi 'detaliile personale ale clientului' (nume, prenume, an nastere, loc nastere, ora nasterii), mail, tipul de client.

Tipul de cont este o clasa enum 'AccountType' si contine urmatoarele tipuri: PERSONAL, BUSINESS.

a. Realizeaza toate clasele, respectand metodele. Adauga niste metode de adaugare, stergere, modificare, intr-un arraylist sau hashmap folosind maparea Map<UUID, TipClasa>
b. Realizeaza o restrictie cu privire la mail-ul fiecarui client. Un mail trebuie sa contina @yahoo.com sau @gmail.com pentru a fi valid si trebuie sa fie de tipul "prenume.nume@gmail.com" sau "prenume.nume@yahoo.com".
c. Realizati o noua functionalitate care are la baza faptul ca un client poate avea mai multe mail-uri. Stocati toate mail-urile si faceti o mapare one-to-many. Un mail nu poate sa corespunda decat la o singura persoana.
d. Realizati o noua functionalitate care are la baza faptul ca, un client are nevoie de un card. Realizati o implementare prin care un client are un singur card si este o relatie one-to-one dintre client si card. Fiecare card are un singur cont, deci avem o relatie one-to-one dintre card si client. Un card are numarul de card, care corespunde cu numarul de cont, CVV (3 cifre), numele si codul pin.
e. Realizati o noua implementare prin care un client poate avea mai multe carduri. Reganditi sistemul pentru a intelege cum se fac maparile si adaugati o constrangere astfel incat numarul cardului este acelasi cu numarul contului, iar numele cardului sa corespunda cu numele clientului.
f. Realizati o noua implementare prin care un client poate face tranzactii. Realizati un sistem prin care la fiecare tranzactie, sa stocam aceste tranzactii. O tranzactie contine informatii precum: data si ora tranzactiei, furnizorul la care a fost facuta tranzactia.
g. Sortati clientii dupa nume, prenume. Adaugati faptul ca clientii pot sa iti activeze sau nu un card. Implementati aceasta functionalitate.
h. Dorim sa facem o sumarizare a informatiei si astfel avem nevoie de o clasa records prin care sumarizam informatiile despre clienti. Clasa records se numeste 'ClientSummary' si stocheaza informatia noastra despre clienti: gruparea pe clienti business si non-business.
i. Sa presupunem ca sistemul nostru are acces la datele furnizorului si astfel dorim sa stocam informatii despre furnizor. Avem informatii despre faptul ca furnizorul nostru are cont la noi, deci este un client business. Realizati un clasament cu cele mai multe tranzactii facute la un furnizor si sa se stocheze informatiile in records-ul definit folosind Collectors.teeing().
j. Realizeaza o sumarizare a performantei si vrem ca dupa ce avem un sumar despre informatiile de mai sus, sa facem o analiza. Pentru asta, trebuie sa folosim un stream peste listele de clienti business si non-business si sa stabilim daca un client a avut tranzactii sau nu. Daca acesta nu a avut tranzactii in ultimele 3 luni, contul si respectiv cardul sa fie dezactivat.
k. Foloseste un stream.anyMatch() pentru verificarea de mai sus pentru a nu itera inutil peste lista. Este posibil ca sa nu existe astfel de clienti.
l. Testeaza fiecare clasa serviciu folosind JUnit.

18. Implementeaza o solutie folosind structura layerizata (model-service, menu singleton class), pentru a implementa un mini-catalog. O sa ai o clasa 'Student' prin care vrem sa stocam informatiile unui student al facultatii, iar clasa 'Profesor' prin care ne definim un profesor al facultatii. Exista relatie many-to-many.

Clasa AbstractEntity este o clasa abstracta care contine id-ul, creationDate, expulsionDate si up-

datedDate.

Clasa Student contine id-ul student-ului, numele de familie, prenumele, id-ul cursurilor la care este inscris, forma de invatamant si daca este student la licenta, master sau doctorat.

Clasa Profesor contine informatii despre profesor: id-ul profesorului, numele de familie, prenume, an nastere, loc nastere, ora nasterii, mail si id-ul materiilor (cursurilor) pe care le preda.

Clasa Catalog contine informatii despre note, deci o sa contina un camp, Map<UUID, PairInfo> prin care mapam id-ul materiei o clasa PairInfo care incapsuleaza la randul ei id-ul studentului cu nota.

Clasa PairInfo trebuie construita ca o clasa care incapsuleaza id-ul studentului cu nota. Clasa Catalog trebuie sa fie construita ca o clasa singleton prin care se da acces la instanta doar printr-o cheie (astfel incat doar profesorii sa poata modifica clasa respectiva. Cheile vor fi stocate intr-o lista statica in clasa Menu, o lista imutabila pe care doar profesorii o pot accesa.

a. Proiecteaza un sistem care sa respecte cerintele de mai sus. Pentru metoda de autentificare creeaza de la tastatura un sistem de logare prin care utilizatorul introduce de la tastatura daca este Student sau Profesor.

b. Realizeaza o noua functionalitate prin care se introduce un serviciu de secretariat. Astfel, implementeaza o clasa 'Secretar' prin care secretarii pot adauga noi studenti in catalog. Profesorii si studentii nu pot adauga noi studenti in catalog.

b-1. Implementeaza o noua functionalitate prin care folosesti un Predicat pentru a stabili daca un profesor, student sau secretar are acces la o metoda.

c. Implementeaza o noua functionalitate prin care sa existe mai multe tipuri de cursuri: optionale, facultative si obligatorii. Grupeaza toti studentii in functie la ce tip de curs sunt. (Bonus: cu un record si extinzand functionalitatea de Collectors.teeing() sa accepte 3 parametrii, nu doar 2. -> este mai greu, trebuie folosit supraincarcarea, extinderea clasei Collectors si preluata implementarea din JVM de la Collectors.teeing() si implementata astfel incat sa accepte 3 argumente si sa se stocheze totul intr-un record numit CourseData)

d. Implementeaza o noua functionalitate prin care limitezi ca un student poate face parte maxim la 2 cursuri optionale si maxim la 2 cursuri facultative, numele si prenumele de familie al profesorilor si studentului trebuie sa inceapa cu litera mare.

d-1. Implementeaza o noua functionalitate prin care un student/profesor sa poata stoca mai multe prenume. Foloseste String.split('-') pentru a afla toate prenumele unui student/profesor si campul care stocheaza prenumele sa fie de tipul 'List<String>'.

d-2. Implementeaza o noua functionalitate prin care poti sa citesti prenumele unui student/profesor invers. Foloseste clasa StringBuilder prin care Poti folosi 'StringBuilder.reverse()'.

e. Implementeaza o noua functionalitate prin care putem face media tuturor studentilor in functie de ce note au in catalog, pe fiecare an de studiu. De retinut este faptul ca un student poate sa-si vada doar media lui, iar un profesor si un secretar pot sa vada toate notele.

f. Implementeaza o noua functionalitate prin care un student poate sa dea mariri. Astfel, implementeaza o metoda prin care un profesor/secretar poate modifica nota din catalog a unui student, doar in sesiunea de mariri/restante. Creeaza o noua clasa Sesiune prin care se stocheaza tipurile de examen (ExamType): EXAMEN_RESTANTA, EXAMEN_MARIRE si EXAMEN_SESIUNE. De retinut este faptul ca un tip de examen incapsuleaza o dificultate. Dificultatea este bazata pe nota pe care studentul o are. Daca studentul a obtinut o nota mai mica (<6.5), atunci dificultatea este mai usoara, iar daca a obtinut o nota mai mare (>=6.5), atunci dificultatea este mai grea. De retinut este si faptul ca in sesiunea de restanta, numarul de probleme este egal cu numarul de probleme din examenul din sesiune, iar in sesiunea de marire, numarul de probleme este cu 2 mai mult. Incapsuleaza in clasa Student examenele pe care le-a sustinut fiecare student printr-un Set<ExamType, Double> prin care

Double este nota examenului, iar ExamType este tipul de examen. Notele din sesiune se stocheaza doar in clasa Sesiune care contine toate notele din sesiune, deci contine un camp static Map<UUID, Set<ExamType, Double> prin care se mapeaza id-ul studentului cu fiecare examen si nota pe care a obtinut-o la examenul respectiv. Implementeaza o metoda pe nume evalueazaStudentii() prin care se atribuie notele obtinute in sesiune tuturor studentilor.

g. Implementeaza o noua functionalitate prin care putem evalua cum au decurs sesiunile. In functie de tipul de examen pe care l-au primit fiecare student, afla media aritmetica a notelor si sorteaza de la cea mai mare la cea mai mica nota. Doar profesorii au acces la aceasta functionalitate.

i. Implementeaza o noua functionalitate prin care fiecare student poate acorda un review fiecarui profesor. Clasa Review contine review-uri ale tuturor studentilor, iar doar studentii pot acorda un review si doar secretarii pot citi review-ul respectiv. Fiecare review contine o intrebare si o nota de la 1 la 5.

i-1. Implementeaza o noua functionalitate prin care daca un profesor are media 1, materia sa fie considerata GREA, iar daca un profesor are media 5, materia sa fie considerata USOARA, iar daca o materie este considerata GREA, aceasta sa se modifice ca fiind facultativa. Se poate folosi '.stream().allMatch()' pe review-uri.

j. Folosind un TreeMap<Float, UUID> care incapsuleaza nota fiecarui profesor si id-ul profesorului, realizeaza un clasament descrescator in functie de nota pe care fiecare profesor a primit-o. Toata lumea poate vedea nota pe care a primit-o profesorul.

k. Testeaza aplicatia folosind JUnit.