READ ME

(tema semestriala 2)

Buta Denisa Elena 322AA

Solutia de ansamblu pe care am ales-o este bazata pe multe dintre metodele invatate atat la curs cat si de-a lungul semestrului (prin teme de laborator,research..) in cadrul disciplinei Programare Orientata pe Obiecte. Pot sa spun ca, fata de ce stiam inainte sa incep semestrul acesta, consider ca am progresat extrem de mult in acest domeniu. Utilitatea temei mi-a fost demonstrata prin cate lucruri am invatat si am consolidat in timp ce o rezolvam. Consider ca implementarea mea este una eficienta, incercand pe cat posibil sa nu incarce inutil codul programului.

IMPLEMENTARE

Am implementat toate enunturile temei.

Specificari/Observatii legate de functiile implementate:

Exercitiul 1

Am implementat constructorii, functiile get si set cerute pentru toate clasele.

Exercitiul 2

Am implementat functiile get si set cerute pentru clasa Server.

Exercitiul 3

a)

Pentru a gasi esspresoarele reduse din lista produselor de pe Server am folosit o lista auxiliara (res) in care am stocat toate elementele care indeplineau conditiile cerute (pentru acestea m-am folosit de functile ajutatoare din clasa Product).

b)

Pentru a gasi toti utilizatorii non-Premium care plateau <=11.5 lei pentru transport, am folosit de asemenea o lista auxiliara in care am stocat toti utilizatorii care indeplineau conditiile respective.

c)

Cum motivul returnarii nu putea fi accesat direct din lista initiala de produse (lista produselor de pe Server fiind de tip Product\*), a fost nevoie de un dynamic cast prin care “converteam” fiecare obiect resigilat din lista intr-unul de tip Non-Food Product\* apoi Returned Product\*(doar pentru cazurile unde se putea face asta, adica pentru obiectele care chiar erau de tip Non-Food Product/Returned Product) apoi le-am adaugat in lista rezultat doar pe acelea care aveau motivul “lipsa\_accesorii”.

Pentru a face sortarea, am implementat in clasa Utility o functie denumita compareNonFoodProductsPrice(dar pentru a putea fi folosita de lista rezultat parametrii trebuiau sa fie de tip Product\*, deci pentru a accesa pretul fiecaruia a fost din nou nevoie de dynamic cast).

d)

Pentru aceasta cerinta am implementat din nou o functie ajutatoare in clasa Utility, compareFoodProducts in care in prima instanta am sortat produsele in functie de nume. Apoi, daca numele era acelasi, am convertit produsele la Food Product\* pentru a putea accesa campurile countryOfOrigin si LeiPerKg si a face o sortare si in functie de acestea.

e)

Aici, pentru prima parte(aceea de a gasii judetul in care locuia numarul maxim de utilizatori) prima data am sortat lista in functie de numele judetului(tot cu o functie in cadrul clasei Utility—am comparat judetul din adresa la care se livra podusul deoarece acolo m-am gandit ca locuieste fiecare utilizator), apoi cu ajutorul unui for am parcurs lista de utilizatori rezultata in urma sortarii.

Am avut nevoie de 2 iteratori pentru a vedea pana unde judetele erau identice. Daca urmatorul utilizator facea parte din acelasi judet ca cel dinainte, acest lucru era contorizat si, daca se depasea un maxim atins la un judet anterior(variabila max ce continea numarul maxim de utilizatori consecutivi ce fac parte din acelasi judet se actualiza doar daca contorul il depasea) aceasta devenea judetul cu numarul maxim de utilizatori.(stocand de asemenea si numele acestuia pentru partea a doua).(k ramane 1 daca nu se indeplineste conditia deoarece exista minim un utilizator in judetul respectiv)

In partea a doua am cautat utilizatorii care faceau parte din judetul respective si locuiau la casa si i-am introdus in lista rezultat pe care ulterior am sortat-o in functie de UserID cu o functie din clasa Utility.

f)

Pentru aceasta cerinta in prima faza am cautat toti utilizatorii premium pe care i-am introdus intr-o lista ajutatoare de tip premiumUser\*(pentru a putea accesa mai tarziu campul discounts).

Apoi, am cautat in map-ul de discount-uri ale fiecarui utilizator premium(folosindu-ne de cheie) produsele ale caror id-uri apareau si in lista de produse de pe server si ale caror categorie era fie imprimanta fie telefon si am introdus in lista rezultat utilizatorii gasiti. Intrucat puteau aparea copii, am folosit functia unique.

Exercitiul 4

Pentru functia processRequests:

Am introdus in lista lru primul obiect din requestsNo deoarece sigur nu era depasita capacitatea si sigur nu se repeat. Apoi, pentru restul obiectelor am avut nevoie de un for care poate fi privit ca avand 2 parti.

Prima parte se refera la situatia de dinaintea umplerii capacitatii cache-ului. Am memorat in vectorul lru(pe prima pozitie) doar elementele din requestsNo care nu apareau deja. Daca se intampla ca un element pe care doream sa il introducem in continuare sa fie identic cu unul deja introdus, il stergeam pe cel identic din lru si introduceam pe prima pozitie elementul din requestsNo.

A doua parte se refera la situatia cand size-ul vectorului lru devenea egal cu capacitatea lui. In acest caz se stergea ultimul element si introduceam pe prima pozitie a lru elementul din requestNo (ca in partea 1—doar daca nu exista deja, caz in care se stergea si se adauga pe prima pozitie si nu mai era nevoie sa stergem ultimul element).

Exercitiul 5

Pentru functiile requestAddProduct requestDeleteProduct am modificat capacitatea produsului doar daca se indeplineau toate conditiile din cerinta.