ROCK the SHOP

Dan Narcis Costinel, Grupa 322AA

Gestiunea propriu-zisa a magazinul este efectuata prin utilizarea unui obiect de tipul Store, astfel se pot retine toate informatiile necesare si se pot utiliza toate metodele, pentru prelucrarea angajatilor si a produselor, dar si realizarea comenzilor si a rapoartelor.

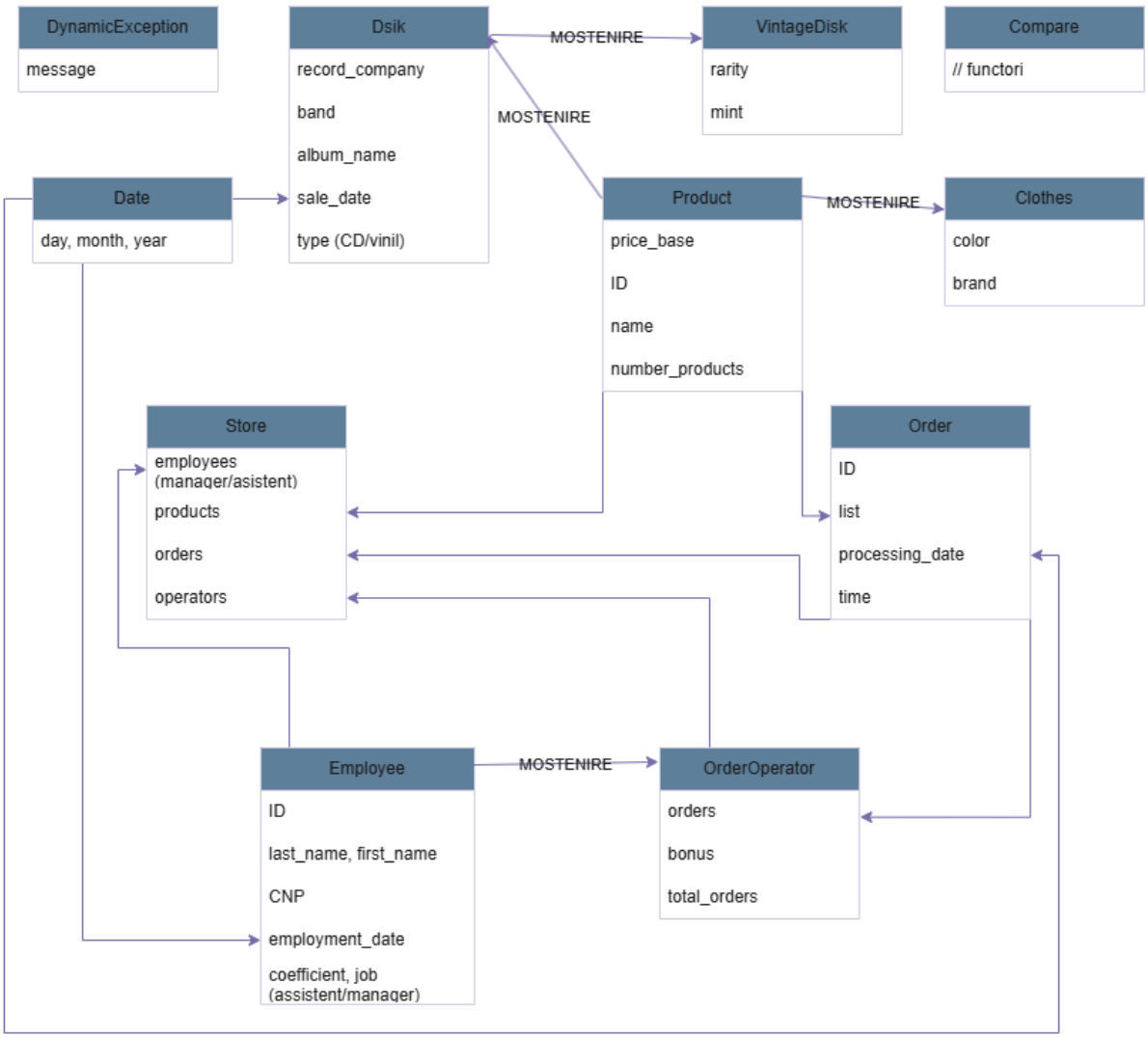
```
class Store {
    vector<shared_ptr<Employee>> employees;
    vector<unique_ptr<Product>> products;
    priority_queue<shared_ptr<OrderOperator>, vector<shared_ptr<OrderOperator>>, Compare> operators;
    priority_queue<unique_ptr<Order>, vector<unique_ptr<Order>>, Compare> orders;

    static unsigned long long orders_processed;
```

Pentru a usura gestiunea comenzilor, angajatii sunt retinuti in 2structuri separate: vector si coada cu prioritati, astfel nu este necesar sa parcurgem vectorul, in cautarea operatorilor, desi am „ingreunat” gestiunea angajatilor, deoarece pentru fiecare operatie este necesar sa folosim 2structuri.

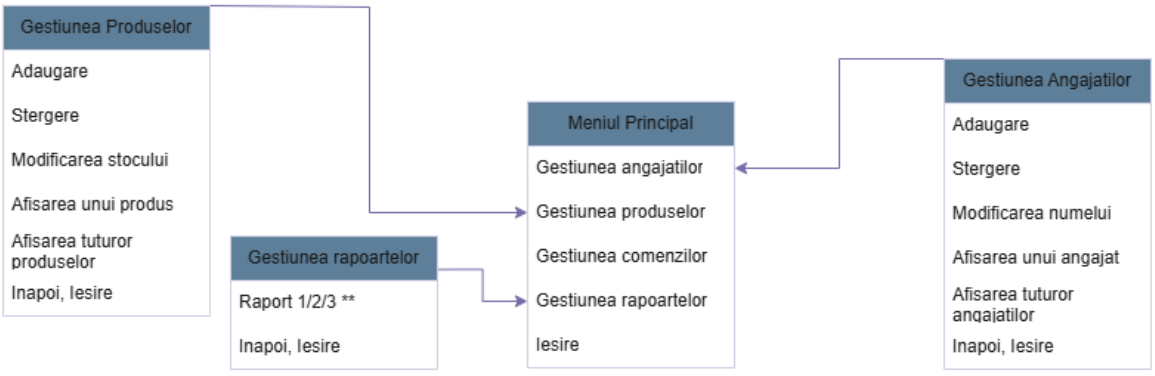
Am folosit shared_ptr pentru angajati, pentru ca folosesc inca o structura cu toti angajatii, pentru a realiza rapoartele, caz, in care este necesar sa am exact 2refetrinte catre acelasi angajat.

In cazul produselor si al comenzilor, nu este necesar sa am mai mult de o referinta catre obiect, un produs sters din vector este eliminat definitiv, iar o comanda, care este exclusa din coada, urmeaza sa fie prelucrata si atribuita operatorului de comenzi, urmand sa fie eliminata definitiv.

IERARHIA CLASELOR



IERARHIA MENIULUI



**

1. Angajatul cu cele mai multe comenzii procesate
2. TOP3 angajati, care au gestionat cele mai valoroase comenzi
3. TOP3 angajati, care au cele mai mari salarii pe luna curenta

NU conteaza ordinea, in care sunt retinute informatiile despre produse si anagajati(manageri & asistenti), DAR conteaza ordinea comenzilor si a operatorilor, pentru eficienta gestiunii.

```
class Compare {
public:
    bool operator()(const shared_ptr<OrderOperator>& elem1, const shared_ptr<OrderOperator>& elem2) {
        return (elem1->ordersNumber() > elem2->ordersNumber());
    }

    bool operator()(const unique_ptr<Order>& elem1, const unique_ptr<Order>& elem2) {
        return (elem1->getTime(false) > elem2->getTime(false));
    }
};
```

Astfel comenzile sunt ordonate crescator dupa durata de realizare, iar operatorii de comenzi sunt ordonati, in functie de disponibilitate (un operator de comenzi nu poate gestiona mai mult de 3comenzi in paralel), capul cozii fiind cel mai disponibil operator, iar ultimul, fiind inaccesibil.

Meniul este unica interactiune a utilizatorului cu aplicatia, acesta trebuie sa introduca informatii valide, altfel se va arunca un mesaj de eroare (mesaj personalizat nume_exceptie: !! eroare !!), de tipul DynamicException si, dupa caz, aplicatia se va reseta, iar utilizatorul este nevoit sa introduca informatiile, de la inceput, pentru a nu produce greseli.

Abstractizare: 1moment de timp = 1iteratie

Pentru realizarea comenzilor se foloseste o bucla for, care se va opri, cand toti operatorii termina de realizat toate comenzile de realizat, acestea vor fi testate (daca pot fi procesate, exista suficiente produse in stoc), inainte sa fie atribuite unui operator, daca nu pot fi realizate, atunci vor fi sterse, daca o comanda poate fi realizata, dar nu exista operatori disponibili, atunci va ramane in coada, pana cand un operator devine disponibil.

Testele facute au constat, atat in a ma asigura ca aplicatia functioneaza, adica operatilor sunt efectuate cu succes, dar si ca utilizatorul este obligat sa introduca informatii corecte, pentru a putea fi prelucrate.

ex. nu pot exista 2angajati cu acelasi cnp/ testatarea functionalitatii functionalitatii magazinului (minim 1manager, 1asistent, 3operatori de comenzi, 2produse din fiecare tip)/ legaturile intre clase au fost realizate corespunzator (agregare & mostenire)/ in cazul mostenirii, daca au fost implementate toate metodele virtuale/ conexiunea intre pagini/ daca utilizatorul doreste za inchida aplicatia, optiunea sa-i fila disponibila, indiferenta pe pagina, pe care se afla...

Probleme intampinate:

Am invatat sa fac o depanare „mai eficienta”, cu ajutorul ei am inceput sa fac teste asupra aplicatiei, dupa ce terminam de facut cateva clase sau ma apucam sa modific codul, incepand din primul fisier, luam fiecare fisier pe rand, pentru a vedea de unde si de ce apar erori.

Am invatat sa creez cozi cu prioritati, utilizand o regula.

Initial, cand m-am apucat sa scriu codul, am implementat totul folosind pointeri normali si, de abia, cand am terminat toata aplicatia, am redeschis cursul si am citit „pointeri inteligenti”... trecerea de la pointeri normali la pointeri inteligenti a fost ceva mai grea decat ma asteptam, pentru ca am utilizat mai mult unique_ptr si trebuia sa tin cont cand si unde am referinta valida, ca sa nu pierd informatiile.