**- TEMA III -**

**SISTEM DE MANAGEMENT A COMENZILOR UNUI DEPOZIT**

Marc Andrei – Aurel

Grupa 30224

Laborator – Tehnici de Programare

Cuprins

1. Obiectivul temei
2. Analiza problemei
3. Proiectare
4. Implementare
5. Rezultate
6. Concluzii
7. Bibliografie

Obiectivul temei

O bază de date reprezintă o modalitate de stocare a unor [informații](https://ro.wikipedia.org/wiki/Informa%C8%9Bie) și [date](https://ro.wikipedia.org/wiki/Dat%C4%83) pe un suport extern (un dispozitiv de stocare), cu posibilitatea extinderii ușoare și a regăsirii rapide a acestora. La prima vedere sarcina poate părea banală. Totuși, în condițiile în care este vorba de a lucra cu milioane de elemente, fiecare putând consta din cantități de date care trebuie accesate simultan prin [Internet](https://ro.wikipedia.org/wiki/Internet) de către mii de utilizatori răspândiți pe întreg globul.

De obicei o bază de date este memorată într-unul sau mai multe [fișiere](https://ro.wikipedia.org/wiki/Fi%C8%99ier). Bazele de date sunt manipulate cu ajutorul [sistemelor de gestiune a bazelor de date](https://ro.wikipedia.org/wiki/Sistem_de_gestiune_a_bazelor_de_date).

Cel mai răspândit tip de baze de date este cel relațional, în care datele sunt memorate în [tabele](https://ro.wikipedia.org/wiki/Tabel). Pe lângă tabele, o bază de date relațională mai poate conține: indecși, proceduri stocate, declanșatori, utilizatori și grupuri de utilizatori, tipuri de date, mecanisme de securitate și de gestiune a tranzacțiilor etc. Relatiile intervin in etapa de modelare logica. In etapa de modelare fizica, tabelele devin fisiere.

Alte tipuri de baze de date sunt modelul ierarhic, modelul orientat pe obiecte și, mai nou, modelul [XML](https://ro.wikipedia.org/wiki/XML).

Limbajul SQL (Structured Query Language) este limbajul utilizat de majoritatea sistemelor de baze de date relationale pentru definirea si manipularea datelor. Mai exista standardul SQL3 care adauga limbajului SQL trasaturi ale modelului obiect-relational. Pentru formularea conceptuala a interogarilor in bazele de date relationale s-au dezvoltat doua limbaje abstracte de interogare: algebra relationala si calculul relational.

**Obiectivul temei** este dezvoltarea unei aplicatii ce proceseaza comenzile unor clienti pentru un depozit. Bazele de date relationale sunt folosite pentru stocarea produselor, clientilor si a comenzilor. Aplicatia trebuie sa fie compusa din urmatoarele clase: clasa de model, care implementeaza modelul de date (produsele, clientii, comenzile), clasa business logic, care contine logica aplicatiei, clasa de prezentare, care contine implementarile ce au legatura cu interfata grafica si clasa de acces la baza de date, care face conexiunea cu baza de date. Se va folosi javadoc pentru documentarea claselor. Javadoc este un generator de documentatii, facut special pentru limbajul de programare Java, pentru generarea de documentatii in format HTML din codul sursa Java. Acest feature nu afecteaza performanta aplicatiei, deoarece toate comentariile sunt ignorate in timpul compilarii, asadar sunt scrise exclusiv pentru intelegerea mai facila a codului, precum si usurarea mentenantei codului.

Analiza problemei

Tema consta in implementarea unui sistem ce stocheaza comenzile unor clienti dintr-un depozit. Pentru a putea stoca informatiile, vom avea nevoie de o baza de date creata. In aceasta baza de date relationala, vom fi nevoiti sa includem, bineinteles, un tabel pentru stocarea comenzilor, dar si pentru stocarea produselor si a clientilor. Pentru a stoca comenzile, clientii si produsele nou introduse, vom avea cate o metoda speciala. In aceste metode, se vor face apeluri de query-uri SQL, de tipul „insert into <table> values <values>”. In urma acestor apeluri, se va crea o noua coloana in tabelul in care se face apelul, cu id auto-generat, ca urmare a constrangerii „auto\_increment” pusa la fiecare cheie primara (id) din fiecare tabel.

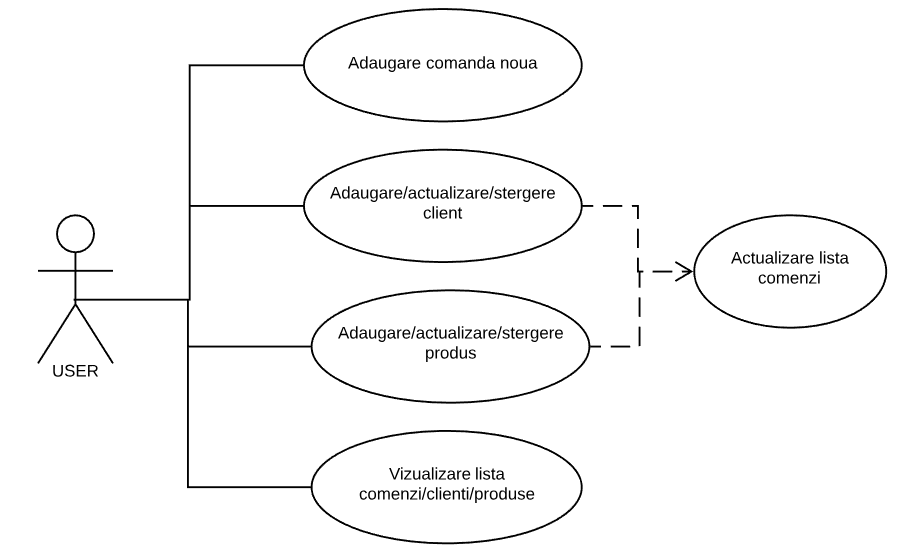
De asemenea, va trebui sa fie disponibila editarea unui produs sau a unui client. Pentru a edita o coloana, trebuie sa ii oferim un camp dupa care sa caute. Am ales ca acest camp sa fie numele. Dupa ce este gasita coloana ce stocheaza informatii despre produsul/clientul cautat, vom apela o metoda ce contine un apel SQL de tipul „update <table> set <column> = ? where <name> = ?”. In urma executiei apelului, se vor schimba campurile date dupa set, cu valorile date.

Pentru a sterge un produs/client, vom folosi o metoda ce contine un apel SQL de tipul „delete from <table> where <id> = ?”. Pentru ca nu putem sti id-ul unui client, vom da ca parametru numele clientului, si vom folosi o metoda ce ii returneaza id-ul, cautand dupa nume. Pentru a sterge un client sau un produs si a evita aparitia unei erori, va trebui sa ii stergem si eventualele comenzi pe care le-a efectuat, pentru ca exista o legatura creata intre cele doua tabele, iar legatura trebuie sa fie constituita din 2 coloane. Daca una din cele 2 coloane este stearsa, atunci legatura ar ramane in aer, iar acest lucru nu este posibil.

Pentru adaugarea unui nou produs/client, am inclus niste butoane care vor declansa aparitia unor field-uri care trebuie completate in vederea adaugarii produsului/clientului nou. Pentru adaugarea unei comenzi, putem sa introducem numele unui client si sa alegem un produs din lista de produse, clientul urmand a fi generat automat, sau putem alege din lista deja creata de clienti. Daca lasam field-ul pentru numele clientului necompletat, se va lua in considerare numele selectat din lista de clienti si se va crea o noua comanda.

Pentru a afisa toate comenzile, vom folosi un JTable, in care vom include coloanele: numarul ordinii, numele clientului, numele produsului, pretul produsului, numarul de produse si pretul total. La adaugarea unei comenzi, va fi automat actualizat tabelul, la fel si in cazul stergerii unei comenzi.

Diagrama Use Case, care prezinta actiunile ce pot fi efectuate de catre un user prin intermediul interfetei.



Un user obisnuit va putea, conform diagramei, sa introduca o comanda noua, un client sau un produs nou. Acestea vor rezulta in actualizarea listei de comenzi. De asemenea, acestuia ii va fi permisa actualizarea sau stergerea unui client sau a unui produs, lucru care va duce, de asemenea, la actualizarea listei de comenzi. Un user va putea vizualiza lista de comenzi, prin intermediul unui JTable, de asemenea si lista de clienti/produse printr-un JComboBox.

Clasa **Client** este clasa care retine informatii despre client, adica numele sau. Are ca metode doar getter-e si setter-e, alte metode nefiind necesare.

Clasa **Product** este clasa care retine informatii despre produse, adica numele, pretul si stocul disponibil al produsului respectiv. Ca metode, are doar getter-e si setter-e.

Clasa **Order** este clasa care retine informatii despre comenzi, adica numeel clientului, produsul si cantitatea de produse comandate. Metodele sunt doar getter-ele si setter-ele necesare pentru atributele definite.

Clasa **BillCreator** este clasa care genereaza o „factura”, un fisier .txt ce contine informatii despre comanda cea mai recenta. In fisier, se va regasi numele clientului si al produsului, pretul produsului, cantitatea comandata si pretul total al comenzii. In coltul din dreapta sus, va fi disponibila si data si ora la care s-a facut comanda.

Clasa **ClientService** este clasa care lucreaza cu informatiile stocate ce au legatura cu clientii, incluzand metode ce modifica, sterg, cauta sau creeaza informatii despre un client.

Clasa **ProductService** este clasa care lucreaza cu informatiile stocate ce au legatura cu produsele. Am inclus metode ce modifica, sterg, cauta sau creeaza informatii despre un produs.

Clasa **OrderService** este clasa care lucreaza cu informatiile stocate ce au legatura cu comenzile efectuate. Include metode ce plaseaza sau cauta o comanda.

Clasa **Database** este clasa care formeaza legatura cu baza de date, pe localhost, portul 3306.

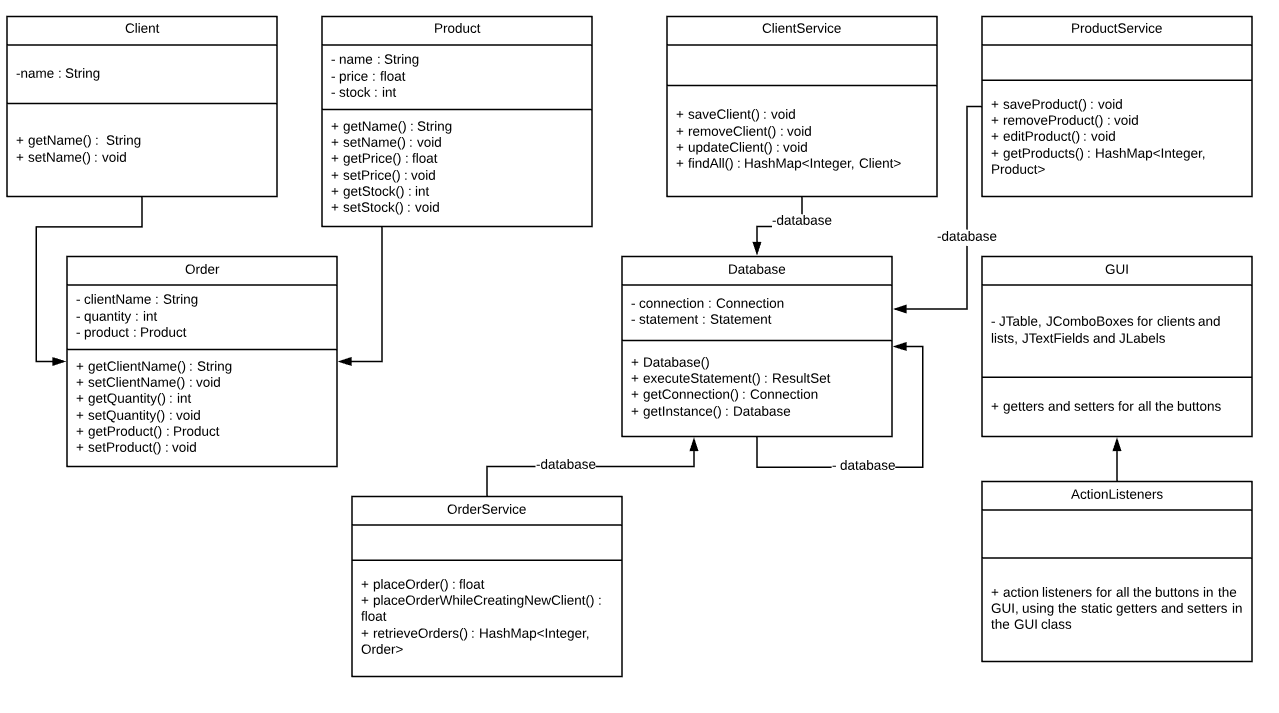
Clasa **GUI** este folosita pentru implementarea interfetei cu utilizatorul. Aceasta contine label-uri ce ofera indicatii despre anumite zone ale interfetei, zonele de text unde sunt date informatiile ce urmeaza a fi stocate in baza de date, precum si lista de produse si clienti, sub forma unor JComboBox, si lista de comenzi, sub forma unui JTable.

Clasa **ActionListeners** este clasa ce contine metode ce implementeaza functionalitatea butoanelor din interfata, adica ce functie vor indeplini ele in momentul in care sunt apasate. Sunt metode statice, insemnand ca nu trebuie creata o instanta a clasei ActionListeners pentru a le putea apela.

Clasele **Validator** sunt folosite pentru validarea informatiilor ce vor fi introduse in baza de date.

Structura proiectului dezvoltat este Maven, un sistem de build și administrare a proiectelor, scris în Java. Proiectul este descris într-un fișier XML, denumit POM (Project Object Model) , care conține informații despre module, precum și despre dependențele proiectului. Acesta îi definește proiectului o denumire unică, formată din group ID și artifact ID.

Diagrama UML



Implementare

Clasa **Client** – folosita pentru stoarea informatiilor despre un client: numele sau, id-ul fiind generat automat in SQL.

* **getName()** si **setName()** sunt getter-ul, respectiv setter-ul numelui clientului

Clasa **Product**  - folosita pentru stocarea informatiilor despre un produs: numele, pretul si stocul disponibil al produsului respectiv, id-ul fiind generat automat in SQL.

* **getName()** si **setName()** sunt getter-ul, respectiv setter-ul numelui produsului
* **getPrice()** si **setPrice()** sunt getter-ul, respectiv setter-ul pretului produsului
* **getStock()** si **setStock()** sunt getter-ul, respectiv setter-ul stocului disponibil al produsului

Clasa **Order** – folosita pentru stocarea informatiilor despre o comanda: numele clientului, produsul comandat si cantitatea de produse comandate.

* **getClientName()** si **setClientName()** sunt getter-ul, respectiv setter-ul numelui clientului
* **getProduct()** si **setProduct()** sunt getter-ul, respectiv setter-ul produsului comandat
* **getQuantity()** si **setQuantity()** sunt getter-ul, respectiv setter-ul cantitatii comandate

Clasa **BillCreator** – folosita pentru crearea unei „facturi”, un fisier text ce contine informatii despre ultima comanda efectuata: data si ora la care s-a efectuat comanda, numele clientului, numele si pretul produsului, numarul de produse si pretul total al comenzii.

* **createBill()** este metoda care genereaza factura, primind ca parametri numele clientului si al produsului, cantitatea si pretul.

Clasa **ClientService** – folosita pentru manipularea datelor din baza de date ce au legatura directa cu tabelul Client. Metodele cele mai importante sunt:

* **saveClient()** este metoda care creeaza un nou client in baza de date
* **removeClient()** este metoda care sterge un client din baza de date, impreuna cu comenzile efectuate
* **updateClient()** este metoda care modifica datele unui client din baza de date. In cazul in care nu se specifica un camp nou, este lasat cel vechi
* **findAll()** este metoda care returneaza un HashMap de clienti, si anume toti clientii inregistrati in baza de date.
* Pe langa aceste metode, se mai implementeaza si alte metode secundare, precum metode care returneaza id-ul clientului dupa un nume dat ca paramentru, si alte metode ce indeplinesc roluri asemanatoare.

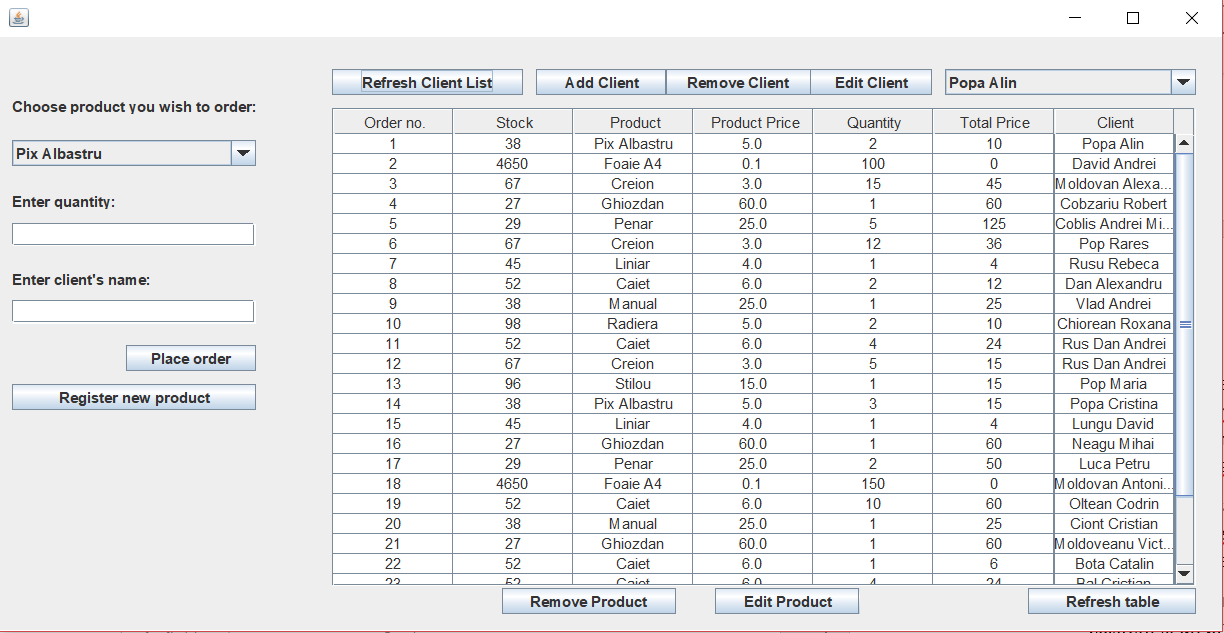
Clasa **ProductService** – folosita pentru manipularea datelor din baza de date ce au legatura directa cu tabelul Product. Metodele cele mai importante sunt:

* **saveProduct()** este metoda care creeaza un nou produs in baza de date
* **removeProduct()** este metoda care sterge un produs din baza de date, impreuna cu toate comenzile efectuate pe produsul efectiv
* **editProduct()** este metoda care modifica un produs din baza de date. In cazul in care un camp nou nu este specificat, cel vechi este lasat
* **getProducts()** este metoda care returneaza un HashMap de produse, si anume totalitatea produselor stocate in baza de date
* Pe langa aceste metode, se mai implementeaza si alte metode ce ajuta la implementarea metodelor principale, ca de exemplu: metoda ce returneaza un produs din baza de date pe baza unui nume dat ca parametru si alte metode ce implementeaza o functionalitate asemanatoare celei descrise mai sus

Clasa **OrderService** – clasa care manipuleaza informatiile stocate ce au legatura directa cu tabelul Order, si anume cu comenzile. Ca si metode principale, aceasta clasa include:

* **placeOrder()** care creeaza o noua comanda, cu un client existent si un produs existent
* **placeOrderWhileCreatingNewClient()** care creeaza o noua comanda, de asemenea creând si un nou client in baza de date
* **retrieveOrders()** care returneaza toate comenzile stocate in baza de date

In clasa **GUI** se implementeaza interfata grafica a aplicatiei, care pune la dispozitie functionalitatile descrise de catre Use Case.



Functionalitatile butoanelor sunt implementate in clasa **ActionListeners**, unde exista cate o metoda pentru fiecare buton.

Interfata este una usor de folosit, fiecare buton avand un nume sugestiv, iar zonele de text ce trebuie completate sunt descrise de label-uri. Spre exemplu, pentru plasarea unei comenzi, se vor completa zonele necesare si se va apasa butonul „Place order”. In cazul in care un nume s-a introdus in text field-ul de sub „Enter client’s name”, se va apela metoda **placeOrderWhileCreatingNewClient()**, altfel se va apela **placeOrder()**, clientul ce plaseaza comanda fiind cel selectat din ComboBox-ul ce contine lista clientilor.

Rezultate

Nu putem avea un cod standard pentru testare, deoarece am crea de mai multe ori acelasi client, insa am testat fiecare pas in implementare pentru a asigura faptul ca implementarea este una corecta. Pentru a testa functionalitatea, am creat o alta clasa, cu o metoda main:

**public** **static** **void** main( String[] args )

{

**new** GUI().setVisible(**true**);

}

Aceasta metoda va afisa interfata si vom putea testa foarte usor functionalitatile proiectului, precum si vizualiza daca operatia efectuata a fost indeplinita.

Concluzii

In aplicatie s-a realizat implementarea diferitelor operatii asupra unei baze de date: modificare, creare, stergere. Baza de date a reprezentat un sistem ce stocheaza datele despre comenzile, clientii si produsele unui depozit. Aplicatia are o structura simpla, fapt ce simplifica dezvoltarea ulterioara sau modificarea anumitor metode.

Ca dezvoltare ulterioara, poate fi luata in considerare posibilitatea de a include mai multe produse pe o comanda.

Bibliografie

<https://www.lucidchart.com/>

<https://stackoverflow.com/>

<https://ro.wikipedia.org/wiki/Baz%C4%83_de_date>