**DOCUMENTATIE**

**TEMA 3**

NUME STUDENT: Mîrza Andrei Tudor

GRUPA: 30224

# CUPRINS

[1. Obiectivul temei 3](#_Toc95297885)

[2. Analiza problemei, modelare, scenarii, cazuri de utilizare 3](#_Toc95297886)

[3. Proiectare 7](#_Toc95297887)

[4. Implementare 8](#_Toc95297888)

[5. Rezultate 9](#_Toc95297889)

[6. Concluzii 9](#_Toc95297890)

[7. Bibliografie 10](#_Toc95297891)

**1.Obiectivul temei**

Obiectivul principal al acestei teme este cel de creare a unei aplicatii de simulare care se pot face asupra unui depozit de produse. Aplicatia va avea 1 mod de functionare, un client un produs si o comanda. Clientul poate sa vizualizeze produse din stock si sa cumpere unul sau mai multe dintre acestea. In momentul in care un client se introduce in baza de date , pentru introducere vom avea nevoie de numele acestuia, adresa , email-ul si numarul de telefon. Iar atunci cand vrem sa adaugam un produs avem nevoie de numele acestuia, pretul si numarul de produse disponibile in stock. Acest simulator este implementat folosind limbajul de programare Java.

Cat despre obiectivele secundare putem spune ca acestea reprezinta pasii care trebuie urmati pentru indeplinirea obiectivului principal si anume:

* Intelegerea cerintelor propuse, analiza problemei
* Proiectarea problemei, scrierea in cod a ideilor de rezolvare
* Testarea aplicatiei

**2.Analiza problemei, modelare, scenarii, cazuri si utilizare**

Analiza problemei – problema este una din viata reala care simuleaza gestionarea unui depozit de produse

Modelarea problemei – este una putin mai complexa intrucat analizand cerinta ne dam seama ca avem nevoie de o baza de date in care sa stocam clientii, produsele si comenzile. Pentru clienti se va retine numele acestora, va avea un id generat automat in baza de date, adresa unde locuieste , email-ul acestuia si numarul de telefon. Pentru produse se va retine numele, pretul si cantitatea, iar pentru comanda avem nevoie de id ul clientului care este generat automat la introducerea acestuia in baza de date , id ul produsului care si acesta este generat automat la introducerea lui in baza de date si cantitatea cumparata.

Cazul in care se adauga un produs:

Utilizator: Angajat

Cazurile de utilizare sunt:

* Angajatorul introduce numele unui produs , pretul acestuia si stocul
* Apasand pe butonul de “Add Product” acesta va fi introdus in baza de date

Cazul in care se sterge un produs:

Utilizator: Angajat

Cazurile de utilizare sunt:

* Angajatorul introduce id ul produsului care va fi sters
* Apasand pe butonul de “Delete Product” produsul va fi sters

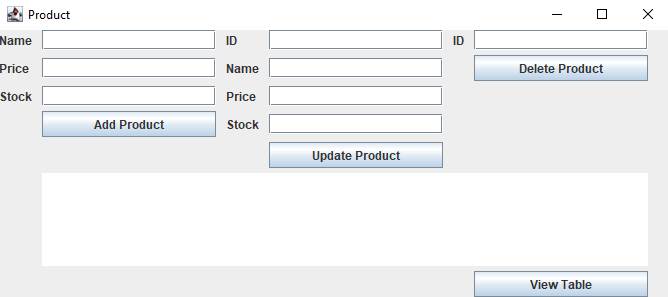
Cazul in care se modifica un produs:

Utilizator: Angajat

Cazurile de utilizare sunt:

* Angajatorul introduce id-ul produsului care urmeaza sa fie modificat, de asemenea se introduce si numele, pretul si numarul de produse aflat in stock

Interfata grafica pentru produse:



Cazul in care se adauga un client:

Utilizator: Angajat

Cazurile de utilizare sunt:

* Angajatorul introduce numele clientului, adresa acestuia, email-ul si numarul acestuia de telefon
* Apasand pe butonul “Add Client” acesta va fi introdus in baza de date cu un id generat automat in ordine crescatoare

Cazul in care se modifica un client:

Utilizator : Angajat

Cazurile de utilizare sunt:

* Angajatorul introduce id ul clientului, numele, adresa , email ul si numarul de telefon
* Apasand pe butonul de “Update Client” , clientul de la id ul adaugat se va modifica conform dorintelor precizate

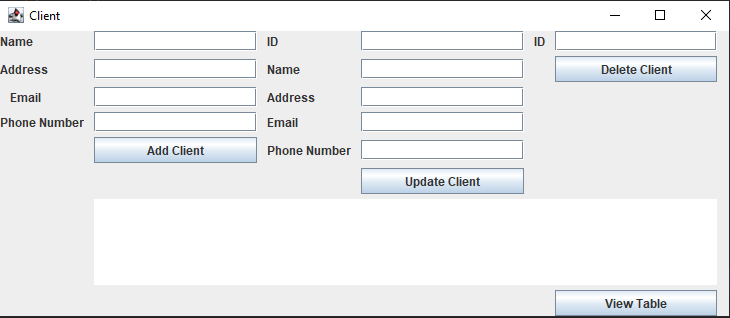
Cazul in care se sterge un client:

Utilizator: Angajat

Cazurile de utilizare sunt:

* Angajatorul trebuie sa introduca doar id-ul clientului care va fi sters
* Apasand pe butonul de “Delete client” clientul va fi sters din baza de date

Interfata grafica pentru client:



Cazul in care se adauga o chitanta:

Utilizator: Angajat

Cazuri de utilizare:

* Angajatul introduce id-ul clientului, id-ul produsului si cantitatea de produse
* Apasand pe butonul de “Add Order” comanda va fi adaugata in baza de date si intr-un fisier “.txt”

Cazul in care se modifica chitanta:

Utilizator:Angajat

Cazuri de utilizare:

* Angajatul introduce id-ul comenzii si cantitatea care urmeaza sa fie modificata
* Apasand butonul “Update Order” comanda se va modifica

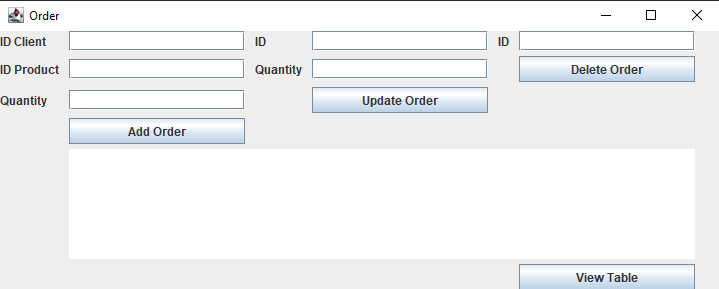
Cazul in care se modifica chitanta:

Utilizator: Angajat

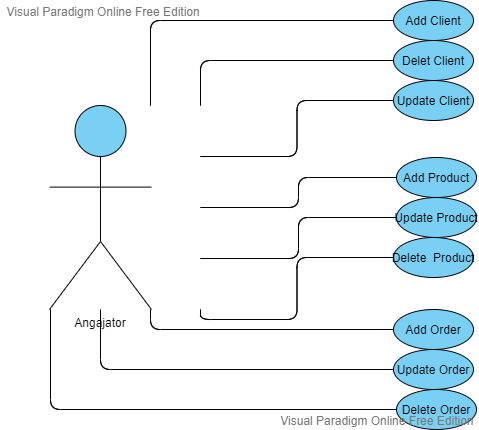
Cazuri de utilizare:

* Angajatul introduce doar id-ul comenzii care va fi stearsa
* Apasand butonul “Delete Order” comanda va fi stearsa

Interfata grafica a comenzii:

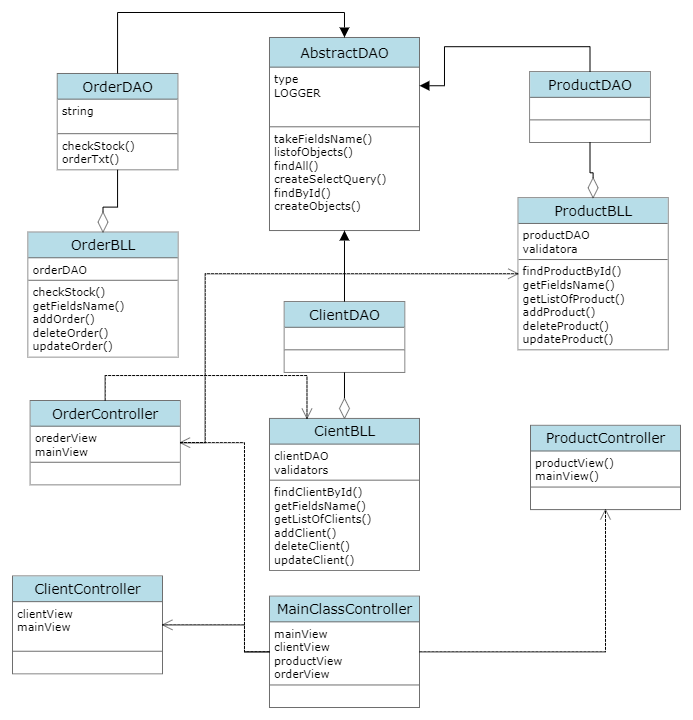


Use case diagram:



**3.Proiectare**

Diagrama claselor:



Aceasta este diagrama de clase si se observa o legatura de model-view-controller.

Dupa cum se poate observa clasa AbstractDAO este clasa parinte pentru clasele ClientDAO, ProductDAO si OrderDAO.Fiecare dintre aceste clase este structurata pe pachete si anume:

* BusinessLogic
* DataAccess
* Model
* Presentation
* Main
* Connnection

BusinessLogic contine clasele : ClientBLL, OrderBLL, ProductBLL

Connection contine clasa ConnectionFactory .

DataAcces contine clasele : AbstractDAO, ClientDAO, OrderDAO, ProductDAO .

Main contine clasa MainClass .

Presentation contine clasele : ClientController, MainClassController, OrderController, ProductController, ClientView, MainView , OrderView si ProductView .

Fiecare dintre aceste clase are anumite metode pe care am sa le prezint in cele ce urmeaza.

**4. Implementare**

Asa cum am zis mai sus fiecare dintre clasele enumerate are mai multe metode care contribuie la realizarea acestei teme.

1. Clasa ClientBLL :

* In aceasta clasa sunt implementate metodele care apeleaza adaugarea unui client, stergerea unui client sau modificarea acestuia
* Pe langa aceste metode mai exista si metoda getFielName() care pune intr-un string numele field-urilor din client dupa care il returneaza
* Tot aici avem si metota getListOfClients() care returneaza o matrice de stringuri avand pe fiecare rand valorile unei liste de obiecte

1. Clasa OrderBLL:

* In aceasta clasa sunt implementate metodele care apeleaza adaugarea unei comenzi addOrder(), stergerea unei comenzi deleteOrder() si modificarea unei comenzi updateOrder()
* Pe langa aceste metode de adaugare, stergere si modificare a unei comenzi mai avem si metoda getFieldNames(), metoda care pune intr – un string numele field – urilor unei comenzi dupa care il returneaza
* tot aici avem si metoda getListOfOrders care returneaza o matrice de string-uri avand pe fiecare rand valorile unei liste de obiecte

1. Clasa ProductBLL

* In aceasta clasa sunt implementate metodele care apeleaza adaugarea unui produs, stergera unui produs sau modificarea acestuia
* Pe langa aceste metode de adaugare, stergere si modificare a unui produs mai avem si metoda getFieldNames(), metoda care pune intr – un string numele field – urilor unui produs dupa care il returneaza

1. Clasa ConnectionFactory

* In aceasta clasa se realizeaza conexiunea cu baza de date

1. Clasa MainClass

* In aceasta clasa sunt apelate clasele MainView, ClientView, ProductView , OrderView, MainClassController, ClientController, ProductController si OrderController

1. Clasa Client:

* In aceasta clasa avem field urile pentru fiecare client plus getter-urile si sett-urile pentru acestea

1. Clasa Orders:

* In aceasta clasa avem la fel ca si in clasa client field – urile pentru o comanda plus getter-urile si sett-urile pentru acestea

1. Clasa Product:

* In aceasta clasa, la fel ca si in clasa Client si Product avem field-urile pentru un produs plus getter-urile si sett-urile pentru acestea

In rest in clasele ClientController, MainClassController, OrderController, ProductController sunt implementate metodele pentru butoanele de adaugarea unui client, stergerea unui client, modificarea unui client, adaugarea unui produs, modificarea unui produs, stergerea unui produs, adaugarea unei comenzi, stergerea unei comenzi si modificarea unei comenzi.

**5.Rezultate**

Datele despre aceste trei entitati trebuie pastrate intr-o baza de date locala si prelucrate prin intermediul unei interfete grafice care se conecteaza la o schema. Pachetele folosite respecta ordinea de management si sunt denumite sugestiv. Fiecare dintre pachete este conectat la Model care reprezinta mapara field-urilor din baze de date in obiecte Java. Clasicul tipar Model –View –Controller este urmat, fiind doar imbunatatit prin divizarea in pahete si limitarea accesarii globale a anumitelor obiecte. Pentru prelucrarea bazei de date s-a folosit cod SQL cu statement-uri clasice (insert, select, update si delete). MySQL Workbench retine local baza de date si printr-un “jar” se asigura conectarea la schema. Pentru acest proiect s-a folosit InteliJ Idea si varianta Java 8 cu librari precum “java.awt”, “java.swing”, “java.lang”, “java.util”,”java.beans” si “java.sql”.

**6.Concluzii**

Principalele lucruri invatate in acest proiect  au fost creare unor usecase-uri mai putin mai complexe , a unor sequence diagram si o mai buna intelegere a legaturilor dintre obiecte si clase fiind obligati sa respectam anumite reguli. In urma implementarii acestei teme , sa dobandit o experienta noua in ceea ce consta programarea , prin folosirea concepului de thread , lucru benefic pentru orice programator ,    s – au recapitulat elemente de interfata grafica incepand de la utilizarea butoanelor , a componentelor JTextField , JTextArea si am invatat sa folosesc JScrollPane pentru a putae vizualiza toate datele de iesire ale proiectului. Un proiect greu daor din punct de vedere a utilzarii thread – urilor , restul programului continand daor elemente invatate pana in acest moment .  Ca o concluzie, acest proiect m-a ajutat sa imi consolidez cunostintele de programare orientata pe obiect dobandite in primul semestru si sa imi organizez munca in baza paradigmelor POO. Totodata,  felul in care am structurat proiectul si am creat clasele si metodele m-a ajutat sa eficientizez codul din punct de vedere al lungimii si al usururintei intelegerii acestuia, lucruri vitale in cazul programelor complexe care necesita mai multe clase sau care presupun un numar mai mare de programatori.

La capitoloul imbunatatiri si dezvoltari ulterioare intra foarte multe elemente . De exempul se poate imbunatati aspectul ferestrei , se pot adauga alte operatii cum ar fi deschiderea caselor si inchiderea caselor in timpul rularii programului , sau deschiderae respective inchiderea acestora in functie de numarul de client.  Tot ca si imbunatatire ar fi crearae unei simulari vizuale in care sa fie prezentate casele si deplasarae clientilor in cozi sau incercarea de a rezolva proiectul prin algoritmi mai eficienti. Evidentierea si folosirea  unor concepte ale programarii orientate pe obiect , introducerea de interfete si/  sau clase abstracete ar putea fi o imbunatatire a codului existent. De asemenea, imbunatatirea interfetei grafice, inchiderea si deschiderea caselor in functie de numarul de clienti, crearea unui algorit mai eficient pentru distribuirea clientilor la case, precum si aducerea de noi posibilitati pentru utilizator,. De asemenea, in ceea ce priveste implementarea claselor si structura proiectului, se poate alege utilizarea unor design patternuri cum ar fi MVC (Model View Controller).

**7.Bibliografie**

1. <https://docs.oracle.com/javase/tutorial/uiswing/>

2.  www.stackoverflow.com

3. https://netbeans.apache.org/kb/docs/java/gui-functionality.html