DOCUMENTATIE

TEMA 3

Student : Mihali Gabriel

Grupa : 30225

Seria : A

Contents

[**1.Obiectivul temei :** 3](#_Toc100679276)

[**2.Analiza problemei, modelare, scenarii, cazuri de utilizare:** 4](#_Toc100679277)

[**3.Proiectare:** 5](#_Toc100679278)

[**4.Implementare:** 6](#_Toc100679279)

[**5.Concluzii:** 11](#_Toc100679280)

[**6.Bibliografie:** 12](#_Toc100679281)

# **1.Obiectivul temei :**

* **Obiectivul principal :** Implementarea unei aplicatii, cu interfata grafica pentru gestionarea unor comenzi si management-ul unui warehouse. Este necesara folosirea bazelor de date de tip relational pentru a stoca clientii , produsele si comenzile.
* **Obiectivele secundare :** 
  + **Crearea claselor Model :** Reprezinta modelele de date pentru aplicatie. Acestea sunt Customer, Product si Order , cu setter-e si getter-e corespunzatoare, si sunt folosite pentru a modela datele query-urilor in Java. Mai multe detalii despre Model in capitolul 4.
  + **Crearea claselor Business Logic :** Aceste clase fac legatura interfetei Java cu partea de SQL din aplicatie. Mai multe detalii despre Business Logic in capitolul 4.
  + **Crearea interfetelor pentru diferitele operatii :** Interfetele vor fi implementate folosind Java Swing, Mai multe detalii despre interfete in capitolul 4.
  + **Crearea claselor DAO (Data Access) :** Cu ajutorul acestor clase accesam datele din baza de date relationala a aplicatiei. Mai multe detalii despre DAO in capitolul 4.
  + **Crearea unor query-uri generice folosind reflection :** Pentru lizibilitatea codului si eficientizarea stilului, s-a optat pentru generalizarea unor metode care creeaza automat query-uri in functie de clasa unde sunt apelate. Mai multe despre reflection in capitolele 3 si 4.
  + **Respectarea arhitecturii Layered :** Mai multe despre arhitectura Layered in capitolul 4.

# **2.Analiza problemei, modelare, scenarii, cazuri de utilizare:**

* **Proiectati si implementati** o aplicatie Orders Management pentru a procesa comenzile unor client intr-un deposit. Baze de date de tip relational vor fi folosite pentru stocarea produselor, a clientilor si a comenzilor. Aplicatia va trebui proiectata conform modelului arhitecturii Layered si ar trebui sa foloseasca(minim) urmatoarele clase :
  + **Clase de tip Model :** modelul de date al aplicatiei
  + **Clase de tip Business Logic :** logica aplicatiei
  + **Clase de tip Presentation :** clase pentru interfata
  + **Clase de tip Data Access :** clase care contin accesul la baza de date
* **Use cases :** 
  + **Actor :** employee
    - **Actions** :

1. Add product
2. Add client
3. Add order
4. Edit product
5. edit client
6. delete product
7. delete client.

* **Actor :** client
  + **Actions :**
    1. Add order

# **3.Proiectare:**

* Aplicatia are 23 clase si un interface, distribuite in 6 pachete, bll, Connection, DataAccess, Model, Presentation si Validators.
  + **Pachetul Model** : Contine 3 clase, clasele dupa care se modeleaza datele, Customer, OrderBill si Product. Acestea au datele de tip private si setter-e si getter-e. Acestea reprezinta fundamentul acestei aplicatii.
  + **Pachetul Connection** : Are o clasa, ConnectionWarehouse, clasa care se ocupa de crearea conexiunii cu baza de date. Are o metoda getter care returneaza conexiunea, de tip DriverManager.
  + **Pachetul DataAcess** : Contine 4 clase, AbstractDAO, CustomerDAO, OrderDAO si ProductDAO. Clasa AbstractDAO este extensa de CustomerDAO, OrderDAO si ProductDAO. Aceste clase contin metode care returneaza date din baza de date, astfel se realizeaza conexiunea intre baza de date si aplicatie.
  + **Pachetul Presentation :** Are 12 clase.Contine clasele pentru interfata grafica, implementata in Java Swing si Controller**-**ul, clasa care leaga toata aplicatia.
  + **Pachetul Validators :** Continue o clasa, EmailValidator si interfata Validator. EmailValidator implementeaza Validator si se ocupa de verificarea unui email valid din input-ul utilizatorului si arunca o exceptie in caz invalid.
  + **Pachetul bll :** Contine 3 clase, CustomerBLL, OrderBLL, ProductBLL, care folosesc clasele din DataAcess pentru a modela niste metode care folosesc clasele din pachetul Model pentru a adauga campuri in baza de date. Astfel se face legatura dintre Model si DataAcess.
  + Ca si **structuri de date speciale**, am folosit StringBuilder pentru a forma un String generic dupa anumiti parametri, am folosit PreparedStatement pentru apelarea query-urilor, si Connection/DriverManager pentru stabilirea conexiunii cu baza de date, ResultSet pentru preluarea datelor din query-urile de tip Select. De asemenea am folosit clasa Object, pentru generalizarea operatiilor si pentru folosirea reflection-ului.
  + Pentru validarea email-ului, am folosit o expresie regex, care verifica toate cazurile unui email gresit. O imagine cu expresia regex folosita este atasata mai jos.

**EXPRESIA REGEX pentru email :**

# **4.Implementare:**

* **Pachetul Model : Se ocupa de modelarea datelor pentru aplicatie :** 
  + - **Clasa Customer :** Reprezinta clientii aplicatiei care vor face comenzi cu anumite produse. Contine metode de getter, getCustomer\_id, getCustomer\_name, getCustomer\_surname, getCustomer\_adress, getCustomer\_email si metode de setter, setCustomer\_id, setCustomer\_name, setCustomer\_surname, setCustomer\_adress si setCustomer\_email. Aceste metode sunt folosite pentru a accesa datele de tip private din clasa. De asemenea avem o metoda toString() care afiseaza date despre clasa Customer. Clasa are 2 constructori, un constructor null, si un constructor care are ca parametri anumite campuri.
    - **Clasa Product :** Reprezinta produsele aplicatiei care vor fi comandate de diversi clienti.Are 2 constructori, un constructor null si un constructor care are ca parametri anumite campuri. Are campurile nume, pret, stoc si id.Are metode de getter si setter si o metoda toString pentru a afisa produsele.
    - **Clasa OrderBill :** Reprezinta comenzile aplicatiei. Contine campuri de tip private int. Campurile customer\_order si product\_order sunt chei straine in baza de date, aceste chei straine se conecteaza de tabelele Customer si Product. Clasa are metode de getter si setter si o metoda toString care afiseaza diverse date despre clasa.
* **Pachetul Connection : Se ocupa de conexiunea cu baza de date :**
  + - **Clasa ConnectionWarehouse :** Are un constructor care initializeaza un Logger si mai multe String-uri care reprezinta date despre conexiune. DBURL reprezinta link-ul spre server-ul bazei de date, user si password reprezinta user-ul si parola bazei de date. Aceasta clasa are o metoda getter statica, care returneaza o metoda din DriverManager cu conexiunea la baza de date. Aceasta clasa este foarte importanta deoarece cu ea se realizeaza conexiunea la baza de date in orice metoda in care se va face un query.
* **Pachetul DataAcess : Se ocupa de accesul datelor din baza de date :** 
  + - **Clasa AbstractDAO :** Are 2 constante, LOGGER si type, si in nume parametrul T. Aceasta clasa contine metode de creare string-uri pentru query-uri generice, de creare a query-urilor generice, de returnare a proprietatilor unei clase si a valorilor campurilor unei clase, prin reflection . Metoda de returnare a proprietatilor unei clase este retrieveProperties care primeste un parametru de tip Object, si adauga intr-o lista de String-uri numele field-urilor din clasa Object-ului, returnate prin metodele getClass() si getDeclaredFields(). Metoda de returnare a valorilor unei clase este retrieveValues care primeste ca parametru un Object si adauga valorile campurilor clasei intr-o lista de tip Object, deoarece valorile pot fi de mai multe tipuri. Folosind getClass() si getDeclaredFields() aceasta metoda initializeaza un Field, din care se va prelua valoarea si se va stoca intr-un alt Object, dupa care, ulterior se va adauga in lista. De asemenea mai contine metoda createObjects, care preia ca parametru un ResultSet, il parcurge si pune intr-o lista de tip T, dat ca parametru la declararea clasei, toate instantele claselor din acel ResultSet. Metodele de formare a string-urilor pentru query sunt : createInsertQuery, createSelectQuery, createUpdateQuery si createDeleteQuery. Acestea iau ca si parametrii liste de field-uri si valori si, folosind type.getSimpleName(), care returneaza numele clasei de catre care este apelata metoda, StringBuilder si append() contruiesc un string in functie de field-uri si valori. Astfel, aceste metode pot fi apelate cu orice field-uri, valori si pe orice tabel, asigurand flexibilitate. Metodele care creeaza query-ul propriu-zis sunt : deleteQuery, updateQuery, insertQuery si viewByName. DeleteQuery creeaza un query de stergere astfel : creeaza un string apeland metoda createDeleteQuery, stabileste conexiunea apeland getConnection() din ConnectionWarehouse, stabileste operanzii query-ului, dati ca parametri in metoda, iar, folosind PreparedStatement, apeleaza query-ul. UpdateQuery modifica un tabel dat cu niste valori noi, updated. Folsoind createUpdateQuery, formeaza un string, pe care il foloseste dupa in query-ul de tip PreparedStatement. Diferenta la aceasta metoda este ca are 2 bucle de parcurgere, pentru 2 tipuri de field-uri : field-uri cu valorile updated si field-urile care reprezinta clasa care trebuie updated. InsertQuery creeaza o instanta noua in tabel. Diferenta in aceaste metoda este ca aceasta verifica daca valorile sunt de tip Integer sau String, pentru a apela corespunzator setInt(), sau setString() pentru parametrii query-ulu. Acest lucru este obtinut prin “instanceof” pe o valoare de tip Object. Metoda viewByName apeleaza un query de tip “SELECT” care returneaza un ResultSet cu instanta din tabel care are field-urile date ca parametru cu valorile date din nou ca parametru.
    - **Clasa CustomerDAO :** Aceasta clasa extinde clasa AbstractDAO, iar in parametrul T pune Customer. Aceasta clasa se foloseste de metodele din AbstractDAO, metode generalizate, pentru a crea metoda mai personalizate pentru datele de tip Customer. Metoda getAllCustomers() apeleaza query-ul “select \* from Customer” si returneaza o lista de Customer prin folosirea metodei createObjects() pe ResultSet-ul din query. Metodele personalizate sunt insertQueryCustomer, viewQueryCustomer, updateQueryCustomer, deleteQueryCustomer. Acestea primesc ca si parametru un obiect de tip Customer, preiau campurile si valorile acelui obiect prin retrieveProperties() si retrieveValues() si modeleaza aceste valori si campuri in functie de cerinta metodei respective. De exemplu, pentru metoda insertQueryCustomer, este ignorat customer\_id din campuri si din valori, deoarece, acesta, la crearea instantei in tabel, se creeaza automat fiind de tip autoincrement.
    - **Clasa OrderDAO :** Ca si CustomerDAO, extinde clasa AbstractDAO, iar in T se pune OrderBill. Are o singura metoda personalizata, createOrderQuery(), care primeste id-ul clientului care face comanda, id-ul produsului, pretul comenzii, si un parametru de tip OrderBill. Apeleaza insertQuery(), ignorand orderID, deoarece acesta este autoincrement.
    - **Clasa ProductDAO :** Ca si OrderDAO, extinde clasa AbstractDAO, iar in T se pune Product. Are mai multe metode personalizate, asemanatoare cu cele din CustomerDAO, cu mici diferente. Diferentele sunt ca in query-urile de update, delete si view, adica in metodele deleteQueryProduct, updateQueryProduct si viewQueryProduct, se ignora campul product\_id.
* **Pachetul Validators :** 
  + - **Clasa EmailValidator :** Foloseste interfata Validator. Este o clasa care implementeaza metoda validate() si valideaza email-ul unui client dat ca input din interfata. Arunca o exceptie de tip IllegalArgumentException daca email-ul nu este valid.
* **Pachetul bll : Clasele din pachetul bll sunt folosite in controller.** 
  + - **Clasa CustomerBLL :** Are 2 campuri, unul de tip Validator, cu care se valideaza datele despre un Customer introduse, si unul de tip CustomerDAO, cu care se realizeaza accesul la baza de date. Contine metodele insertCustomer(), viewCustomer(), deleteCustomer(), updateCustomer() si getCustomer() care primesc parametri de tip Customer si apeleaza metodele din customerDAO.
    - **Clasa OrderBLL :** Are un camp, de tip OrderDAO, cu care se realizeaza accesul la baza de date, si o singura metoda createOrder, care ia ca parametru parametrii de la metoda createOrderQuery din orderDAO si o apeleaza.
    - **Clasa ProductBLL :** Are un camp de tip ProductDAO, cu care se realizeaza accesul tabelei Product din baza de date. Contine metodele insertProduct(), viewProduct(), deleteProduct(), updateProduct() si getProducts() care primesc parametri de tip Product si apeleaza metodele din ProductDAO.
* **Pachetul Presentation : Contine clasele pentru interfata grafica : AddCustomerView, AddProductView, CustomerDetails, EditCustomerView, EditProductView, EditSaveCustomerView, EditSaveProductView, MainView, OrderView, ProductDetails, TableView. O imagine cu interfata MainView este atasata mai jos. De asemenea contine clasa Controller.** 
  + - **Clasa Controller** : Din clasa Controller se va porni aplicatia. Aceasta are ca si campuri toate interfetele, clasele din pachetul bll, clasele din Model si conexiunea la baza de date. Aceasta clasa implementeaza ActionListener-ele pentru butoanele din interfete, formand clase din pachetul Model si apeland diverse metode din clasele BLL in functie de butonul apasat. De asemenea, clasa Controller arunca si diferite mesaje de eroare in functie de input-ul introdus de catre utilizator in interfete. De asemenea, clasa Controller genereaza niste fisiere de tip .txt cu detaliile comenzilor.

**In figura de mai jos este prezentat package diagram-ul aplicatiei :**

**Diagram

Description automatically generated**

**In figura de mai jos este prezentata diagrama package-ului Model:**

**Table

Description automatically generated with medium confidence**

**In figura de mai jos este prezentata interfata MainView :**

**Graphical user interface

Description automatically generated**

**Interfata OrderView, care se ocupa de comenzi :**

**Graphical user interface

Description automatically generated**

# **5.Concluzii:**

* Aceasta aplicatie a aprofundat cunostintele despre MySQL, despre implementarea unei aplicatii care lucreaza cu query-uri de tip SQL, despre conexiunile bazelor de date cu aplicatiile. De asemenea au fost asimilate cunostinte noi despre reflection, despre generalizarea metodelor si a claselor. Au fost asimilate cunostinte noi despre arhitectura Layered si despre Business Logic.
* Ca si dezvoltari ulterioare, poate fi implementat un sistem de login pentru utilizatori cu interfata, ca selectarea clientului, cand se plaseaza o comanda, sa se faca automat.

# **6.Bibliografie:**

<https://gitlab.com/utcn_dsrl/pt-reflection-example>

<https://gitlab.com/utcn_dsrl/pt-layered-architecture>

<https://docs.oracle.com/javase/7/docs/api/java/io/FileWriter.html>

<https://www.w3schools.com/sql/default.asp>  
https://jenkov.com/tutorials/java-reflection/index.html