

DOCUMENTATIE

TEMA 3

NUME STUDENT: BOTANEL MIRONA
GRUPA: 30224

CUPRINS

1.	Obiectivul temei.....	3
2.	Analiza problemei, modelare, scenarii, cazuri de utilizare	3
3.	Proiectare.....	3
4.	Implementare.....	5
5.	Rezultate.....	Error! Bookmark not defined.
6.	Concluzii	10
7.	Bibliografie.....	10

1. Obiectivul temei

Principal: Proiectarea și implementarea unei aplicații pentru gestionarea comenzilor clienților pentru un deposit

Secundar:

-Analiza problemei și identificarea cerințelor

-Proiectarea aplicației de gestionare a comenzilor

-Implementarea aplicației de gestionare a comenzilor-Testați aplicația de gestionare a comenzilor

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Proiectul e conceput a fi o aplicație de management a comenzilor, clienților și produselor, care utilizează o interfață grafică pentru a interacționa cu utilizatorii și o bază de date pentru a stoca și gestiona datele.

Cerintele sunt:

Requirement	Grading
<ul style="list-style-type: none">• Use an object-oriented programming design, classes with maximum 300 lines, methods with maximum 30 lines, Java naming conventions.• Use <i>javadoc</i> for documenting classes and generate the corresponding JavaDoc files.• Use relational databases for storing the data for the application, minimum three tables: Client, Product and Order.• Create a graphical user interface including:<ul style="list-style-type: none">• A window for client operations: add new client, edit client, delete client, view all clients in a table (JTable)• A window for product operations: add new product, edit product, delete product, view all product in a table (JTable)• A window for creating product orders - the user will be able to select an existing product, select an existing client, and insert a desired quantity for the product to create a valid order. In case there are not enough products,	5 points

<p>an under-stock message will be displayed. After the order is finalized, the product stock is decremented.</p> <ul style="list-style-type: none">• Use reflection techniques to create a method that receives a list of objects and generates the header of the table by extracting through reflection the object properties and then populates the table with the values of the elements from the list.• Good quality documentation covering the sections from the documentation template.	
Layered Architecture (the application will contain at least four packages: dataAccessLayer, businessLayer, model and presentation).	2 points
Define an immutable Bill class in the Model package using Java records. A Bill object will be generated for each order and will be stored in a Log table. The bills can only be inserted and read from the Log table; no updates are allowed.	1 point
Use reflection techniques to create a generic class that contains the methods for accessing the DB (all tables except Log): create object, edit object, delete object and find object. The queries for accessing the DB for a specific object that corresponds to a table will be generated dynamically through reflection.	2 points

3. Proiectare

• STRUCTURI DE DATE

-ArrayList: Folosit în toate clasele pentru a stoca o listă de obiecte de tipul Validator. De asemenea, este folosit pentru a stoca lista de ID-uri ale clienților în ClientDAO.

-List: Interfața este folosită pentru declarațiile variabilelor și returnările din metode, iar implementarea concretă este ArrayList.

-StringBuilders: StringBuilders sunt folosite pentru a construi interogări SQL dinamic. Sunt folosite în metode precum createSelectQuery() și createInsertQuery() pentru a construi interogări SELECT și INSERT.

-Arrays: Arrays sunt folosite pentru a stoca date temporar, cum ar fi ID-urile recuperate din baza de date. Acestea sunt apoi convertite în liste pentru manipulare mai ușoară. Acest lucru este văzut în metode precum `getIds()`.

-ResultSets: ResultSets sunt folosite pentru a recupera date din baza de date după executarea interogărilor SQL. Sunt utilizate în metode precum `findById()`, `findAll()` și `createObjects()` pentru a prelua date din baza de date și a le mapa la obiecte.

-Modelul JTable (DefaultTableModel): Este folosit pentru a furniza date componente JTable în interfața grafică. Stocază datele într-un format tabular.

- **ALGORITMI FOLOSITI**

-Algoritmi de Acces la Bază de Date:

Pentru a accesa și manipula datele din baza de date, am folosit algoritmi de interogare a bazelor de date, cum ar fi SELECT, INSERT, UPDATE și DELETE, în combinație cu JDBC (Java Database Connectivity).

-Algoritmi de Validare a Datelor:

Am implementat algoritmi pentru validarea datelor introduse de utilizatori în interfața grafică, inclusiv verificări pentru cantități, nume de clienți și adrese de email valide.

-Algoritmi pentru Manipularea Obiectelor Java:

Am folosit algoritmi pentru crearea, actualizarea, ștergerea și căutarea obiectelor Java, cum ar fi `findById`, `findAll`, `insert`, `update` și `delete`.

-Algoritmi pentru Interfața Grafică:

Am implementat algoritmi pentru gestionarea interacțiunii utilizatorului cu interfața grafică, inclusiv logica de afișare a ferestrelor, manipularea evenimentelor utilizatorului și actualizarea interfeței grafice în funcție de acțiunile utilizatorului.

-Algoritmi pentru Manipularea Șirurilor de Caractere:

Am utilizat algoritmi pentru manipularea șirurilor de caractere, cum ar fi concatenarea, divizarea și formatarea, pentru a prelucra și afișa datele în mod corespunzător în interfața grafică.

-Algoritmi de Tratare a Excepțiilor:

Am implementat algoritmi pentru gestionarea excepțiilor, inclusiv capturarea, înregistrarea și gestionarea excepțiilor care pot apărea în timpul rulării aplicației.

-Algoritmi de Reflexie:

Am utilizat algoritmi de reflexie pentru a inspecta și accesa dinamic proprietățile obiectelor Java, precum și pentru a afișa informații despre acestea.

- **DIAGRAMA DE CLASE**



4. Implementare

Se va descrie fiecare clasa cu campuri si metodele importante. Se va descrie implementarea interfetei utilizator.

- *ReflectionExemple*

Se utilizează reflecția pentru a recupera și afișa proprietățile (câmpurile) unui obiect.

- *Bill*
- *Client*

Reprezintă entitatea unui client. Corespunde cu tabelul *Client* din *MySQL*. Analog pentru clasele *Product*, *Orders* si *Bill*

- *MainFrame*

Reprezintă interfața grafică principală a aplicației. Ea conține trei butoane care permit utilizatorului să acceseze diferite funcționalități ale aplicației: comenzi (*Orders*), clienți (*Clients*) și produse (*Products*). Atunci când un buton este apăsat, este afișată o fereastră corespunzătoare (*OrdersFrame*, *ClientsFrame* sau *ProductsFrame*) în funcție de acțiunea selectată. Această clasă folosește un design simplu și stilizat pentru butoanele sale și organizează elementele într-un mod vizual plăcut, facilitând navigarea utilizatorului în cadrul aplicației.

Main Menu

Orders

Clients

Products

Orders Management

ID	Client ID	Product ID	Quantity
1	2	2	1
2	3	3	1
3	4	4	1
4	5	5	1
5	6	6	1
6	7	7	1
7	8	8	1
10	11	11	5
11	1	13	2
12	17	13	5
13	17	13	5


Client ID:

Product ID:

Quantity:

Add

Back

 Clients Management

ID	Name	Email	Address
17	miro	miro@yahoo.rom	camin
18	vlad	vlad@yahoo.comic	cluj
19	cata	cata@tahoo.com	sebes
20	dragos	dragos@yahoo.com	sebes
21	mircea	miry@yahoo.com	tg jiu

Name:

Email:

Address:

Add

Delete

Update

Find By ID

Get All IDs

Back

Products Management

ID

Name

Price

Quantity

7	Laptop	9	1500
15	telefon	1200	6
16	tableta	3020	3
17	Laptop	110	10

Name:

Price:

Quantity:

Update

Find By ID

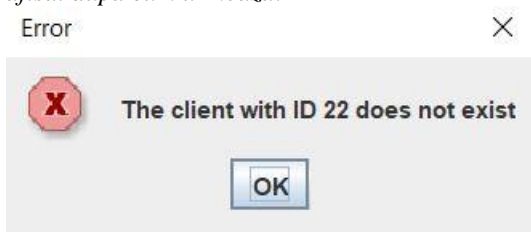
Get All IDs

Back

Add

Delete

In cazul in care trebuie afisat un mesaj de eroare, de exemplu, Nu exista Clientul cu ID-ul 22, mesajul va fi afisat dupa cum urmeaza:



- **AbstractDAO**

Clasa *AbstractDAO* furnizează o implementare generică pentru obiectele de acces la date (DAO) utilizate în interacțiunea cu baza de date. Această clasă abstractă este parametrizată cu un tip generic *T*, care reprezintă entitatea cu care se lucrează în baza de date.

-Gestionarea Conexiunii:

Clasa utilizează *ConnectionFactory* pentru a obține conexiunea la baza de date și pentru a închide conexiunile, declarațiile și rezultatele seturilor de rezultate.

-Operațiuni CRUD:

Metodele *findById*, *findAll*, *insert*, *update* și *delete* oferă operații CRUD (Create, Read, Update, Delete) pentru entitățile din baza de date.

Aceste metode permit găsirea unei entități după ID, găsirea tuturor entităților, inserarea unei noi entități, actualizarea unei entități existente și ștergerea unei entități după ID.

-Generare Automată de Interogări SQL:

Clasa generează automat interogări SQL pentru diferite operațiuni, cum ar fi selecția, inserarea și actualizarea, folosind metodele private createSelectQuery, createInsertQuery și construind interogările pe baza metadatelor entităților.

-Manipularea Reflecției:

Clasa utilizează reflecția Java pentru a accesa și a seta câmpurile entităților, permitând astfel operarea asupra entităților fără a cunoaște detalii specifice despre acestea.

-Înregistrarea Evenimentelor:

Clasa utilizează un obiect Logger pentru a înregistra evenimente și mesaje de avertizare asociate operațiunilor cu baza de date.

- **BillDAO**

Clasa BillDAO furnizează metode specifice pentru manipularea entităților de tip factură (Bill) în baza de date.

Funcționalități Principale:

-Metoda findAll:

Recuperează toate facturile din baza de date și le încarcă într-o listă.

Utilizează o interogare SQL simplă pentru a selecta toate coloanele din tabela Bill.

-Metoda findByIdOrder:

Găsește o factură după ID-ul comenzii asociate.

Utilizează o interogare SQL parametrizată pentru a selecta o factură bazată pe ID-ul comenzii.

-Metoda insertBill:

Inserează o nouă factură în baza de date.

Utilizează o interogare SQL parametrizată pentru a insera valorile câmpurilor unei facturi în tabela Bill.

-Metodele update și delete:

Aceste metode sunt suprascrise din clasa AbstractDAO, dar sunt aruncate excepții

UnsupportedOperationException deoarece actualizarea și ștergerea facturilor nu sunt acceptate.

-Închiderea Resurselor:

Utilizează clasa ConnectionFactory pentru a obține și a închide conexiunea la baza de date, precum și declarațiile și rezultatele seturilor de rezultate.

-Utilizarea Logger-ului:

Utilizează un obiect Logger pentru a înregistra mesaje de avertizare asociate operațiunilor cu baza de date.

Această clasă oferă metode esențiale pentru gestionarea operațiunilor CRUD legate de entitățile de tip factură, permițând astfel interacțiunea cu baza de date într-un mod eficient și sigur.

- **ClientDAO, OrdersDAO, ProductDAO**

extind clasa AbstractDAO<Client> și oferă metode specifice pentru manipularea entităților de tip client, order sau product în baza de date.

- **ClientBLL**

Această clasă gestionează operațiile de bază și complexe legate de entitățile de tip client și oferă un nivel de abstractizare între nivelurile de interfață și de acces la date, facilitând astfel dezvoltarea și întreținerea aplicației.

- **OrderBLL**

Această clasă asigură gestionarea eficientă și logică a operațiilor legate de comenzile din sistem și asigură coerența datelor între diferitele entități din cadrul aplicației.

- **ProductBLL**

Această clasă asigură gestionarea eficientă și logică a operațiilor legate de produsele din sistem și asigură coerența datelor între diferitele entități din cadrul aplicației. De asemenea, oferă feedback utilizatorului prin afișarea mesajelor de eroare în cazul în care operațiile nu pot fi efectuate cu succes.

- **Main**

Clasa Main este punctul de intrare al aplicației. Aici, sunt inițializate și pornite toate componentele necesare pentru funcționarea aplicației.

5. Concluzii

Am învățat să dezvolt o aplicație Java care să gestioneze o bază de date și să interacționeze cu utilizatorul printr-o interfață grafică.

Am dobândit cunoștințe despre utilizarea JDBC pentru a conecta și interoga o bază de date MySQL dintr-o aplicație Java.

Am înțeles importanța validării datelor introduse de utilizator pentru a asigura integritatea datelor din baza de date.

Am învățat să implementez diverse operații CRUD (Create, Read, Update, Delete) pentru a gestiona entitățile din baza de date.

Posibile dezvoltari:

Autentificare și Autorizare: Implementarea unui sistem de autentificare și autorizare pentru diferite nivele de utilizatori, cu permisiuni specifice.

Interfață Grafică Avansată: Dezvoltarea unei interfețe grafice mai complexe și mai intuitive, care să ofere funcționalități suplimentare și o experiență mai plăcută utilizatorilor.

6. Bibliografie

<https://mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/>

<https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>

<https://www.baeldung.com/javadoc>

<https://jenkov.com/tutorials/java-reflection/index.html>