Documentație tema 3.
Gestionarea bazelor de date.

1. Obiectivul temei

Obiectivul acestei teme este de a crea o interfață grafică prin care utilizatorul poate interacționat cu o bază de date a unui magazin. Utilizatorul trebuie să poată vedea clienții magazinului, informații despre acestea cum ar fi, numele, adresa, adresa de email, vârsta, precum și să adauga clienți noi sau modifica respectiv șterge clienți existenți. De asemenea utilizatorul poate gestiona produsele din magazin, adăuga produse noi, a edita sau a șterge produse existente, cât a și interoga stocul magazinului, ce produse există și cantitatea acestora. În plus utilizatorul are disponibilitatea de a crea comenzii prin selectarea unui client dintre clienții existenți ai magazinului și de a selecta un produs existent în magazin, comanda se stochează atât în baza de date cât și într-un fișier generat pentru fiecare comandă plasată.

2. Analiza problemei

Aplicația se folosește pentru a realiza operații pe o bază de date, așadar aplicația trebuie să comunice cu baza de date. Se folosește o baza de date de tip relațional MySQL și s-a ales driverul JDBC pentru a facilita comunicarea între programul Java și baza de date MySQL.

Fiecare dintre facilitățile puse la dispoziție de aplicație reprezintă diferite use-case-urilor cu diferite diferențe între ele.

Use case 1: Introducerea unui nou client

Actorul principal: Utilizatorul aplicației

Principalul scenariu de succes:

1. Utilizatorul introduce în interfața grafică datele noului client în câmpurile destinate.
2. Utilizatorul apasă pe buton de creare a unui nou client.
3. Se citesc datele noului client și se convertesc în reprezentarea internă a unui client.
4. Se introduce noul client generat în baza de date a clienților.
5. Se afișează un mesaj de succes.

Scenariu alternativ:

1. Utilizatorul introduce date care nu sunt valide pentru un client.
2. Se afișează ca datele nu sunt corecte în interfața grafică și nu se realizează nicio modificare la nivelul bazei de date.

Use case 2: Editarea unui client existent

Actorul principal: Utilizatorul aplicației

Principalul scenariu de succes:

1. Utilizatorul introduce în interfața grafică numele clientului asupra căruia se dorește realizarea modificării și doar datele care se doresc a fi modificate.
2. Utilizatorul apasă pe buton de editare a unui client existent.
3. Se citesc noile date ale client și se convertesc în reprezentarea internă a unui client.
4. Se caută în baza de date clientul cu numele specificat și se modifică datele existente.
5. Se afișează un mesaj de succes.

Scenariu alternativ 1:

1. Utilizatorul introduce date care nu sunt valide pentru un client.
2. Se afișează ca datele nu sunt corecte în interfața grafică și nu se realizează nicio modificare la nivelul bazei de date.

Scenariu alternativ 2:

1. Utilizatorul introduce un nume care nu există în baza de date.
2. Se afișează că nu s-a produs nicio modificare în baza de date deoarece nu există clientul cu numele specificat.

Use case 3: Ștergerea unui client existent

Actorul principal: Utilizatorul aplicației

Principalul scenariu de succes:

1. Utilizatorul introduce în interfața grafică numele clientului și eventual informații adiționale despre client.
2. Utilizatorul apasă pe buton de ștergere a unui client existent.
3. Se citesc noile date ale client și se convertesc în reprezentarea internă a unui client.
4. Se caută în baza de date clienții cu numele specificat și informațiile adiționale daca au fost specificate.
5. Se șterg toți clienții care îndeplinesc toate condițiile specificate de ștergere
6. Se afișează un mesaj de succes.

Scenariu alternativ 1:

1. Utilizatorul introduce date care nu sunt valide pentru un client.
2. Se afișează ca datele nu sunt corecte în interfața grafică și nu se realizează nicio modificare la nivelul bazei de date.

Scenariu alternativ 2:

1. Utilizatorul introduce un nume care nu există în baza de date.
2. Se afișează că nu s-a produs nicio modificare în baza de date.

Use case 4: Afișarea clienților existenți

Actorul principal: Utilizatorul aplicației

Principalul scenariu de succes:

1. Utilizatorul apasă pe butonul de afișare a tuturor clienților existenți în baza de date.
2. Se afișează un tabel în interfața grafică cu toți clienții existenți și datele despre aceștia.

Use case 5: Introducerea unui nou produs

Actorul principal: Utilizatorul aplicației

Principalul scenariu de succes:

1. Utilizatorul introduce în interfața grafică datele noului produs în câmpurile destinate.
2. Utilizatorul apasă pe buton de creare a unui nou produs.
3. Se citesc datele noului produs și se convertesc în reprezentarea internă a unui produs.
4. Se introduce noul produs generat în baza de date a produselor.
5. Se afișează un mesaj de succes.

Scenariu alternativ:

1. Utilizatorul introduce date care nu sunt valide pentru un produs.
2. Se afișează ca datele nu sunt corecte în interfața grafică și nu se realizează nicio modificare la nivelul bazei de date.

Use case 6: Editarea unui produs existent

Actorul principal: Utilizatorul aplicației

Principalul scenariu de succes:

1. Utilizatorul introduce în interfața grafică numele produsului și noua cantitate pentru produsul care se dorește a fi modificat.
2. Utilizatorul apasă pe buton de editare a unui produs existent.
3. Se citesc noile date ale produsului și se convertesc în reprezentarea internă a unui produs.
4. Se caută în baza de date produsul cu numele specificat și se modifică cantitatea existentă.
5. Se afișează un mesaj de succes.

Scenariu alternativ 1:

1. Utilizatorul introduce date care nu sunt valide pentru un produs.
2. Se afișează ca datele nu sunt corecte în interfața grafică și nu se realizează nicio modificare la nivelul bazei de date.

Scenariu alternativ 2:

1. Utilizatorul introduce un nume care nu există în baza de date.
2. Se afișează că nu s-a produs nicio modificare în baza de date deoarece nu există produsul cu numele specificat.

Use case 7: Ștergerea unui produs existent

Actorul principal: Utilizatorul aplicației

Principalul scenariu de succes:

1. Utilizatorul introduce în interfața grafică numele produsului și cantitatea produsului.
2. Utilizatorul apasă pe buton de ștergere a unui produs existent.
3. Se citesc noile date ale produsul și se convertesc în reprezentarea internă a unui produs.
4. Se caută în baza de date produsele care îndeplinesc cele 2 criterii.
5. Se șterg toți clienții care îndeplinesc cele 2 criterii.
6. Se afișează un mesaj de succes.

Scenariu alternativ 1:

1. Utilizatorul introduce date care nu sunt valide pentru un produs.
2. Se afișează ca datele nu sunt corecte în interfața grafică și nu se realizează nicio modificare la nivelul bazei de date.

Scenariu alternativ 2:

1. Utilizatorul introduce un nume care nu există în baza de date.
2. Se afișează că nu s-a produs nicio modificare în baza de date.

Use case 8: Afișarea produselor existente

Actorul principal: Utilizatorul aplicației

Principalul scenariu de succes:

1. Utilizatorul apasă pe butonul de afișare a tuturor produselor existente în baza de date.
2. Se afișează un tabel în interfața grafică cu toate produsele existente și cantitatea lor.

Use case 9: Crearea unei comenzi

Actorul principal: Utilizatorul aplicației

Principalul scenariu de succes:

1. Utilizatorul alege un client dintre clienții existenți și un produs dintre produsele existente și introduce o cantitate.
2. Se citesc datele alese și introduse și se convertesc în reprezentarea internă a unei comenzi.
3. Se introduce noua comandă în baza de date.
4. Se actualizează stocului produsul ales.
5. Se creează un fișier text în care sunt scrise datele comenzii.
6. Se afișează un mesaj de succes.

Scenariu alternativ:

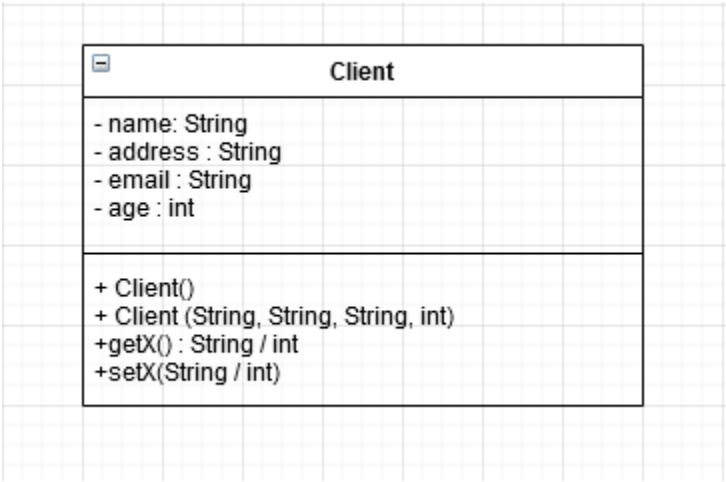
1. Utilizatorul introduce o cantitate care depășește cantitatea disponibilă a produsul.
2. Se afișează că nu există cantitatea cerută.

3. Proiectare

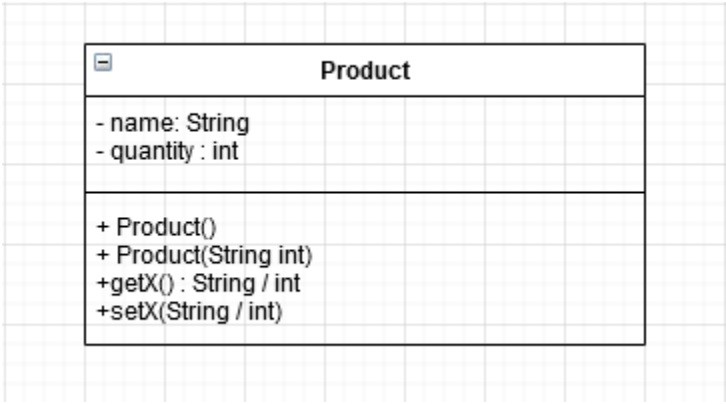
Se folosește **Layered Architecture** care împarte aplicația în 4 pachete importante. Pachetul **model** care conține clasele care reprezintă datele, clienții, produsele respectiv comenzile. Pachetul **presentation** în care se află implementarea interfeței grafice și a controller-ului acesteia. Pachetul **dataAccessLayer** în care sunt clasele pentru gestiunea bazei de date și pachetul **businessLayer** în care sunt implementate clase auxiliare pentru prelucrarea datelor.

Pachetul **model:**

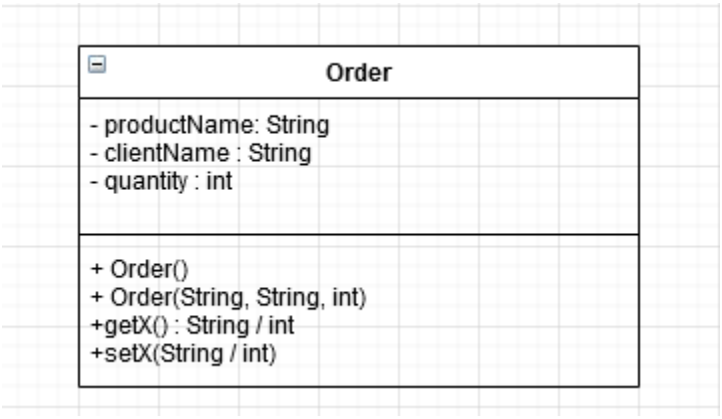
1. Este alcătuit din clasa **Client** care mapează informațiile unui client, cum ar fi numele, adresa, adresa de email și vârsta, constructori pentru crearea noilor clienți și metode de set și get aferente câmpurilor.



2. Clasa **Product** care mapează informațiile unui produs, numele produsului și cantiatea disponibilă din acesta.



3. Clasa **Order** care mapează informațiile unei comenzi, numele produsului, numele clientului și cantitatea comandată.

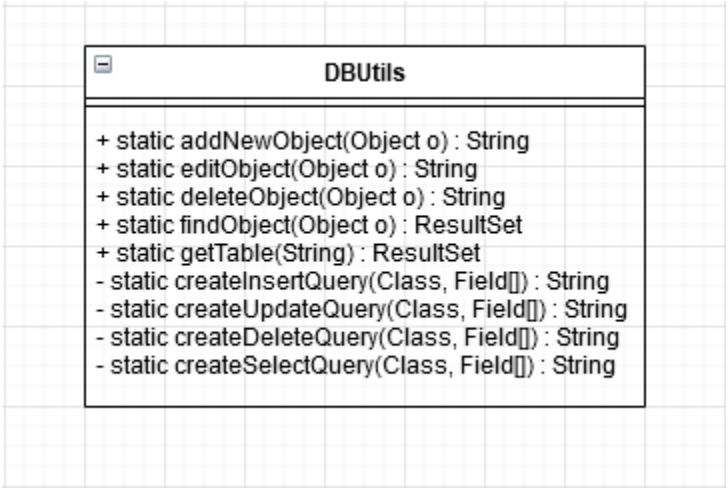


Pachetul **presentation:**

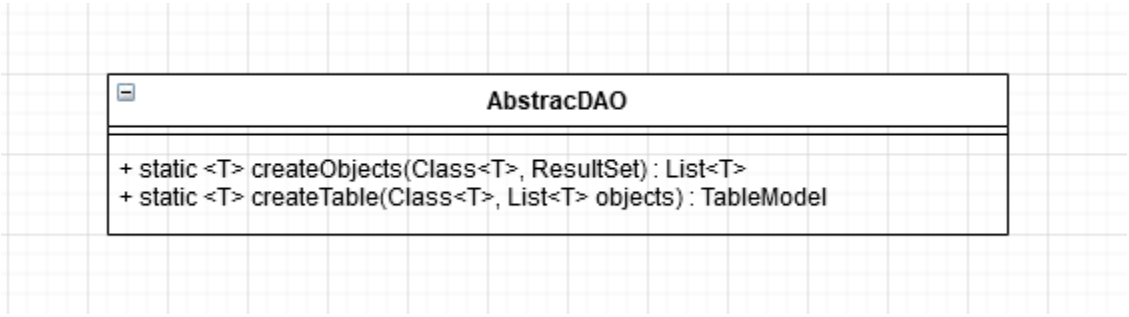
1. Clasa **View –** implementează interfața grafică
2. Clasa **Controller –** creează toți listenerii butoanelor din interfața grafică și realizează legaturile între toate clasele

Pachetul **dataAccesLayer:**

1. Clasa **ConnectionFactory –** are ca scop crearea conexiunii cu baza de date
2. Clasa **DBUtils –** clasa implementează principalele operații pe baza de date, adaugare, editare, ștergere și căutare. Toate operațiile sunt generice, în funcție de obiectul primit.

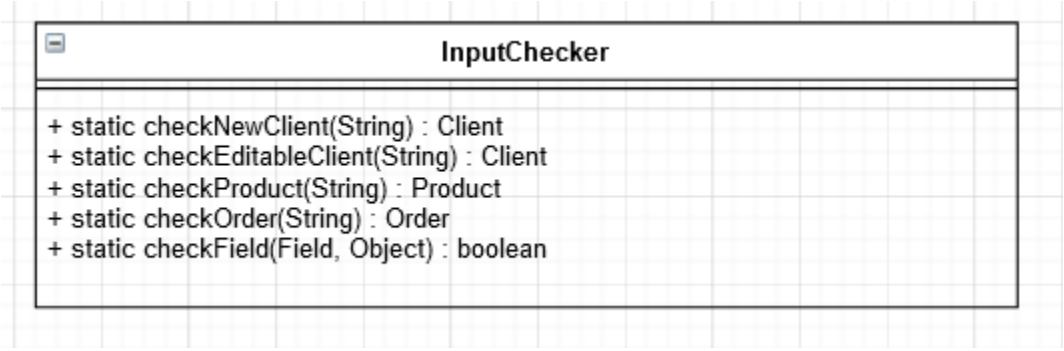


3. Clasa **AbstractDAO** – clasa implementează operațile generice de creare de obiecte și crearea de tabel în funcție de tipul obiectelor pe care le primește.



Pachetul **businessLayer:**

1. Clasa **InputChecker –** pune la dispoziție methode prin care se verifică dacă datele introduse în interfața grafică sunt corecte, dacă sunt corecte returnează un obiect de tipul specificat.



2. Clasa **BillCreator** – care creează pentru fiecare comandă un fișier cu informațiile comenzii.

4. Implementare

Pachetul **model:**

Clasa **Client:**

Variabile:

- String name – numele clientului
- String address – adresa clientului
- String email – adresa de email a clientului
- Int age – vârsta clientului

Metode:

- Client() – constructor gol, folosit pentru instanțierea obiectelor prin reflecție
- Client(String, String, String, int) – constructor pentru crearea obiectelor de tip client

- String/int getField() – metode ce returnează starea variabilelor de instanță
- void setField(String / int) – metode care setează starea variabilelor de instanță
- String toString() – descrierea obiectului sub forma de șir de caractere

Clasa **Product:**

Variabile:

- String name – numele produsului
- int quantity – cantitatea produsului

Metode:

- Product() – constructor gol, folosit pentru instanțierea obiectelor prin reflecție
- Product(String, int) – constructor pentru crearea obiectelor de tip produs
- String/int getField() – metode ce returnează starea variabilelor de instanță
- void setField(String / int) – metode care setează starea variabilelor de instanță
- String toString() – descrierea obiectului sub forma de șir de caractere

Clasa **Order:**

Variabile:

- String productName – numele produsului
- String clientName – numele clientului
- int quantity – cantitatea comandată

Metode:

- Order() – constructor gol, folosit pentru instanțierea obiectelor prin reflecție
- Order(String, String, int) – constructor pentru crearea obiectelor de tip comandă
- String/int getField() – metode ce returnează starea variabilelor de instanță
- void setField(String / int) – metode care setează starea variabilelor de instanță
- String toString() – descrierea obiectului sub forma de șir de caractere

Pachetul **presentation:**

Clasa **View:**

Variabile:

- Componente javax.swing pentru crearea interfeței grafice

Metode:

- void changeView(String viewName) – modifică interfața grafică cu interfața corespunzătoare selecției (client/product/order)
- String[] wrapXXInputFields() – construiește un array de șiruri de caractere care sunt intrările din interfață aferente ferestrei selectate și le returnează Controller-ului pentru a le prelucra (XX – Client/Product/Order)
- void setTable(TableModel, int) – setează conținutul tabelului cu indexul dat ca argumentul
- void printLog(String, int) – afisează în zona de scriere a ferestrei specificate ca parametru un mesaj

- void setAvailableClientsAndProductsToOrder(String[], String[]) – setează selecțiile posibile in fereastra Order in funcție de clienții și produsele existente în baza de date, primește ca argumente numele acestora
- ... – methode pentru crearea interfeței grafice

Clasa **Controller:**

- Implementează clase listener pentru butoanele din interfața grafică, și face legăturile între toate clasele și metodele acestora pentru a deservi comanda primită de la buton.

Variabile:

- view – referintă la interfața grafică

Pachetul **dataAccessLayer:**

Clasa **ConnectionFactory:**

- Clasa de tip singleton

Variabile:

- static connection – conexiunea la baza de date
- informații despre baza de date și conexiunea cu aceasta

Metode:

- Connection getConnection() – returnează conexiunea la baza de date
- void close(Connection/Statement/ResultSet) – metode de inchidere
- Connection createConnection() – creează conexiunea

Clasa **DBUtils:**

- Toate metodele primesc ca parametru un Object îi stabilește tipul folosind reflecția și îl prelucrează în concordanță cu acesta.

Metode:

- String addNewObject(Object) – primește ca parametru un obiectul pe care îl introduce în baza de date în funcție de tipul acestuia, returnează un mesaj despre starea execuției
- String editObject(Object) – primește ca parametru un obiectul pe care îl cauta în baza de date în funcție de tipul acestuia și nume și modifică cu noile date datele găsite, returnează un mesaj despre starea execuției
- String deleteObject(Object) – primește ca parametru un obiectul pe care îl caută în baza de date în funcție de tipul acestuia și șterge obiectele găsite, returnează un mesaj despre starea execuției
- ResultSet findObject(Object) – primește ca parametru un obiectul pe care îl caută în baza de date în funcție de tipul acestuia, returnează un ResultSet cu obiectele găsite
- ResultSet getTable(String) – returnează toate obiectele din tabelul primit ca argument
- String createXXQuery(Class, Filed[]) – creează un query in funcție de operația pe care dorim să o efectuăm și tipul obiectului (XX – Insert/Update/Delete/Select)

Clasa **AbstractDAO:**

- Dispune de metode generice care folosind tipul primit și introspecția obiectelor primite ca parametru, pentru prelucrarea obiectelor

Metode:

- <T> List<T> createObjects(Class<T>, ResultSet) – creează o listă de obiecte de tipul primit ca parametrul din ResultSet
- <T> TableModel createTable(Class<T>, List<T>) – creează un tabel din obiectele primite ca argumentul în funcție de tipul acestora

Pachetul **businessLayer:**

Clasa **InputChecker:**

- Dispune de metode pentru vereficarea intrarilor din interfața grafică

Metode:

- Client checkClient(String[]) – verifica dacă este un client valabil de creat
- Client checkEditableClient(String[]) – verifica dacă este un client valabil de editat
- Client checkProduct(String[]) – verifica dacă este un produs valabil
- Client checkOrder(String[]) – verifica dacă este o comandă valabilă
- Boolean checkField(Field, Object) – verifică daca field-ul primit ca parametrul pentru obiectul primit ca parametrul reprezintă un field valabil pentru o interogare

Clasa **BillCreator:**

Variabile:

- dateTimeFormatter – stabilește formatul timpului

Metode:

- void createBill(Order) – creează un fișier text cu informațiile comenzii

5. Rezultate

S-au testat diferite tipuri de input pentru fiecare dintre comenzile posibile și modul în care reacționează aplicația la acestea, precum și toate combinațiile posibile între inputurile din interfața grafică. S-a asigurat ca nu este nicio excepție care să nu fie prinsă de aplicație și să termine rularea acesteia brusc. În cazul în care se aruncă o excepție din cauza datelor de intrare sau a altei probleme aplicația afisează pe interfața grafică mesajul acesteia si permite utilizatorului să introducă comenzi noi.

6. Concluzii

Din această temă s-a învățat cât de puternică este tehnica de reflecție și cum poate fi folosită pentru a trata obiectele într-o maniera mai generala și în funcție de tipul acestora la runtime precum și flexibilitatea oferită de această metodă.

O dezvoltarea ulterioară ar fi o interfață grafică mai potrivită cât și implementarea mai multor metode de managment a bazelor de date prin aplicația dezvoltată.

7. Bibliografie

https://stackoverflow.com

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

https://docs.oracle.com/javase/tutorial/uiswing/components/

https://www.w3schools.com