DOCUMENTAȚIE

TEMA *3*

NUME STUDENT: Coșarcă Ioan-Cristian

GRUPA: …30227…

CUPRINS

1. Obiectivul temei

Se va prezenta obiectivul principal al temei printr-o frază și un tabel sau o listă cu obiectivele secundare. Obiectivele secundare reprezintă pașii care trebuie urmați pentru îndeplinirea obiectivului principal. Fiecare obiectiv secundar va fi descris și se va indica în care capitol al documentației va fi detaliat.

Obiectivul acestui proiect / acestei teme este realizarea în Java a unei aplicații cu interfață utilizator dedicată care să se ocupe de gestionarea mai multor comenzi date de către utilizator. Aplicația va reține detalii despre Studenții / Clienții inserați, produsul selectat și datele comenzii. Aplicația va semnala de asemenea dacă o anumită comandă este prea mare și nu poate fi ca atare realizată.

Sub-obiective:

- Analiza problemei, modelare, scenarii, cazuri de utilizare (Capitolul 2)
- Proiectare (Capitolul 3)
- Implementare (Capitolul 4)
- Rezultate (Capitolul 5)

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

Se va prezenta cadrul de cerințe funcționale formalizat și cazurile de utilizare ca și diagrame și descrieri de use-case. Descrierile use-case-urilor se vor face sub forma unui flow-chart ori sub forma unei liste conținând pașii execuției fiecărui use-case.

Cerințe funcționale:

- Aplicația de gestiune de comenzi ar trebui să îi permită utilizatorului să introducă un Client / Student în baza de date, împreună cu datele sale personale
- Aplicația de gestiune de comenzi ar trebui să îi permită utilizatorului să introducă un Produs în baza de date, împreună cu informațiile despre acesta
- Aplicația de gestiune de comenzi ar trebui să îi permită utilizatorului să introducă o comandă în baza de date, împreună cu informațiile despre aceasta
- Aplicația de gestiune de comenzi ar trebui să permită modificarea informațiilor unui Client / Student și a unui Produs de către utilizator, aceste modificări rămânând în baza de date
- Aplicația de gestiune de comenzi ar trebui să permită ștergerea unui Client / Student sau a unui produs din baza de date de către utilizator
- Aplicația de gestiune de comenzi ar trebui să poată să afișeze în cadrul interfeței grafice un anumit Client / Student, Produs sau o anumită Comandă, căutate după un ID specificat
- Aplicația de gestiune de comenzi ar trebui să poată afișa în cadrul interfeței grafice o evidență a tuturor Clienților / Studenților, Produselor și Comenzilor existente la acel moment în cadrul bazei de date

Cerințe non-funcționale:

- Aplicația de gestiune de comenzi ar trebui să fie intuitivă și ușor de folosit de către utilizator
- Aplicația de gestiune de comenzi ar trebui să permită utilizatorului să selecteze pe ce tabel dorește să efectueze operația curentă dintre Student / Client, Product și Orders
- Aplicația de gestiune de comenzi ar trebui să ofere suport pentru a afișa pe ecran, în ferestre diferite, conținutul a cel puțin două tabele în același timp

Use Case: Inserare Client / Student

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul inserează în interfața grafică valorile pentru: nume, adresă, email și vârstă.

2. Utilizatorul apasă pe butonul “INSERT”.

3. Aplicația va valida în spate datele introduse, iar dacă sunt valide, respectivul Client / Student va fi inserat în tabel.

Secvență alternativă: Valori invalide pentru datele Clientului / Studentului

- Utilizatorul inserează o adresă de email nevalidă sau o vârstă prea mare sau prea mică pentru Client / Student.
- Datele vor fi verificate de validatoare și se vor arunca excepții aferente.
- Clientul / Studentul respectiv nu va fi inserat în baza de date.
- Se revine la pasul 1.

Use Case: Actualizare date Client / Student cu Nume dat

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul inserează în interfața grafică noi valori pentru: adresă, email și vârstă; și va specifica un nume existent în baza de date a Clientului / Studentului a cărui date se dorește a fi modificate.

2. Utilizatorul apasă pe butonul “UPDATE”.

3. Aplicația va valida în spate datele introduse, iar dacă sunt valide, informațiile respectivului Client / Student sunt actualizate în tabel.

Secvență alternativă: Valori invalide pentru datele Clientului / Studentului

- Utilizatorul inserează o adresă de email nevalidă sau o vârstă prea mare sau prea mică pentru Client / Student.
- Datele vor fi verificate de validatoare și se vor arunca excepții aferente.
- Clientul / Studentul respectiv nu va fi inserat în baza de date.
- Se revine la pasul 1.

Use Case: Ștergere Client / Student după un nume dat

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul specifică în interfața grafică numele Clientului / Studentului pe care dorește să îl șteargă din baza de date.

2. Utilizatorul apasă pe butonul “DELETE”.

3. Clientul / Studentul cu numele specificat va fi șters.

Secvență alternativă: ID-ul specificat nu există în baza de date

- Utilizatorul inserează un nume al unui Client / Student care nu există în baza de date.
- Înregistrările din tabel vor rămâne nemodificate.
- Se revine la pasul 1.

Use Case: Căutare Client / Student după un ID dat

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul specifică în interfața grafică ID-ul Clientului / Studentului pe care dorește să îl caute și a cărui informații dorește să le vizualizeze din baza de date.

2. Utilizatorul apasă pe butonul “FIND ID”.

3. Clientul / Studentul cu ID-ul specificat va fi afișat de interfață.

Secvență alternativă: ID-ul specificat nu există în baza de date

- Utilizatorul inserează un ID al unui Client / Student care nu există în baza de date.
- Se va arunca o excepție și nu se va afișa nimic.
- Se revine la pasul 1.

Use Case: Căutare Client / Student după un nume dat

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul specifică în interfața grafică numele Clientului / Studentului pe care dorește să îl caute și a cărui informații dorește să le vizualizeze din baza de date.

2. Utilizatorul apasă pe butonul “FIND NAME”.

3. Clientul / Studentul cu numele specificat va fi afișat de interfață.

Secvență alternativă: Numele specificat nu există în baza de date

- Utilizatorul inserează un nume al unui Client / Student care nu există în baza de date.
- Se va arunca o excepție și nu se va afișa nimic.
- Se revine la pasul 1.

Use Case: Afișarea tuturor Clienților / Studenților din baza de date

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul apasă pe butonul “VIEW ALL”.

2. Interfața grafică va afișa toți Clienții / Studenții prezenți în cadrul tabelului.

Use Case: Inserare Produs

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul inserează în interfața grafică valorile pentru: nume, preț și stoc.

2. Utilizatorul apasă pe butonul “INSERT”.

3. Aplicația va valida în spate datele introduse, iar dacă sunt valide, respectivul Produs va fi inserat în tabel.

Secvență alternativă: Valori invalide pentru datele Produsului

- Utilizatorul inserează valori negative pentru stocul produsului sau valori mai mici sau egale cu 0 pentru preț.
- Datele vor fi verificate de validatoare și se vor arunca excepții aferente.
- Produsul respectiv nu va fi inserat în baza de date.
- Se revine la pasul 1.

Use Case: Actualizare date Produs cu Nume dat

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul inserează în interfața grafică noi valori pentru: preț și stoc; și va specifica un nume existent în baza de date a Produsului a cărui date se dorește a fi modificate.

2. Utilizatorul apasă pe butonul “UPDATE”.

3. Aplicația va valida în spate datele introduse, iar dacă sunt valide, informațiile respectivului Produs sunt actualizate în tabel.

Secvență alternativă: Valori invalide pentru datele Produsului

- Utilizatorul inserează valori negative pentru stocul produsului sau valori mai mici sau egale cu 0 pentru preț.
- Datele vor fi verificate de validatoare și se vor arunca excepții aferente.
- Produsul respectiv nu va fi inserat în baza de date.
- Se revine la pasul 1.

Use Case: Ștergere Produs după un nume dat

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul specifică în interfața grafică numele Produsului pe care dorește să îl șteargă din baza de date.

2. Utilizatorul apasă pe butonul “DELETE”.

3. Produsul cu numele specificat va fi șters.

Secvență alternativă: Numele specificat nu există în baza de date

- Utilizatorul inserează un nume al unui Client / Student care nu există în baza de date.
- Înregistrările din tabel vor rămâne nemodificate.
- Se revine la pasul 1.

Use Case: Căutare Produs după un ID dat

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul specifică în interfața grafică ID-ul Produsului pe care dorește să îl caute și a cărui informații dorește să le vizualizeze din baza de date.

2. Utilizatorul apasă pe butonul “FIND ID”.

3. Produsul cu ID-ul specificat va fi afișat de interfață.

Secvență alternativă: ID-ul specificat nu există în baza de date

- Utilizatorul inserează un ID al unui Produs care nu există în baza de date.
- Se va arunca o excepție și nu se va afișa nimic.
- Se revine la pasul 1.

Use Case: Căutare Produs după un nume dat

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul specifică în interfața grafică numele Produsului pe care dorește să îl caute și a cărui informații dorește să le vizualizeze din baza de date.

2. Utilizatorul apasă pe butonul “FIND NAME”.

3. Produsul cu numele specificat va fi afișat de interfață.

Secvență alternativă: Numele specificat nu există în baza de date

- Utilizatorul inserează un nume al unui Produs care nu există în baza de date.
- Se va arunca o excepție și nu se va afișa nimic.
- Se revine la pasul 1.

Use Case: Afișarea tuturor Produselor din baza de date

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul apasă pe butonul “VIEW ALL”.

2. Interfața grafică va afișa toți Clienții / Studenții prezenți în cadrul tabelului.

Use Case: Creare Comandă

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul selectează din interfața grafică numele unui Client / Student, numele unui Produs și introduce o valoare pentru cantitate.

2. Utilizatorul apasă pe butonul “Place Order”.

3. Aplicația va valida în spate datele introduse, iar dacă sunt valide, informațiile comenzii respective vor fi inserate în tabel. De asemenea, se va actualiza stocul produsului cu numele specificat.

Secvență alternativă: Valori invalide pentru datele Comenzii

- Utilizatorul inserează o cantitate prea mare în comparație cu stocul disponibil al produsului.
- Datele vor fi verificate de validatoare și se vor arunca excepții aferente.
- Comanda respectivă nu va fi inserată în baza de date, iar tabela Produselor va rămâne neatinsă.
- În cadrul interfeței, într-o zonă text, se va afișa un mesaj informativ.
- Se revine la pasul 1.

Use Case: Căutare Comandă după un ID dat

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul specifică în interfața grafică ID-ul Comenzii pe care dorește să o caute și a cărei informații dorește să le vizualizeze din baza de date.

2. Utilizatorul apasă pe butonul “View by ID”.

3. Comanda cu ID-ul specificat va fi afișată de interfață.

Secvență alternativă: ID-ul specificat nu există în baza de date

- Utilizatorul inserează un ID a unei Comenzi care nu există în baza de date.
- Se va arunca o excepție și nu se va afișa nimic.
- Se revine la pasul 1.

Use Case: Afișarea tuturor Comenzilor din baza de date

Actor Primar: Utilizatorul

Scenariu de succes:

1. Utilizatorul apasă pe butonul “View All Orders”.

2. Interfața grafică va afișa toate Comenzile prezente în cadrul tabelului.

3. Proiectare

*Se va prezenta proiectarea OOP a aplicației, diagramele UML de clase și de pachete, structurile de date folosite, interfețele definite și algoritmii folosiți (*dacă e cazul)

Proiectarea aplicației de gestiune de comenzi respectă arhitectura Layered Architecture. Aceasta se împarte în 4 categorii:

- Model
- Presentation Layer
- Business Layer
- Data Access Layer

Modelul conține clasele mapate cu baza de date.

Presentation Layer cuprinde clasele care definesc interfața utilizator.

Business Layer conține clasele care încapsulează logica aplicației.

Data Access Layer conține clasele ce cuprind interogările tabelelor și conexiunile cu baza de date.

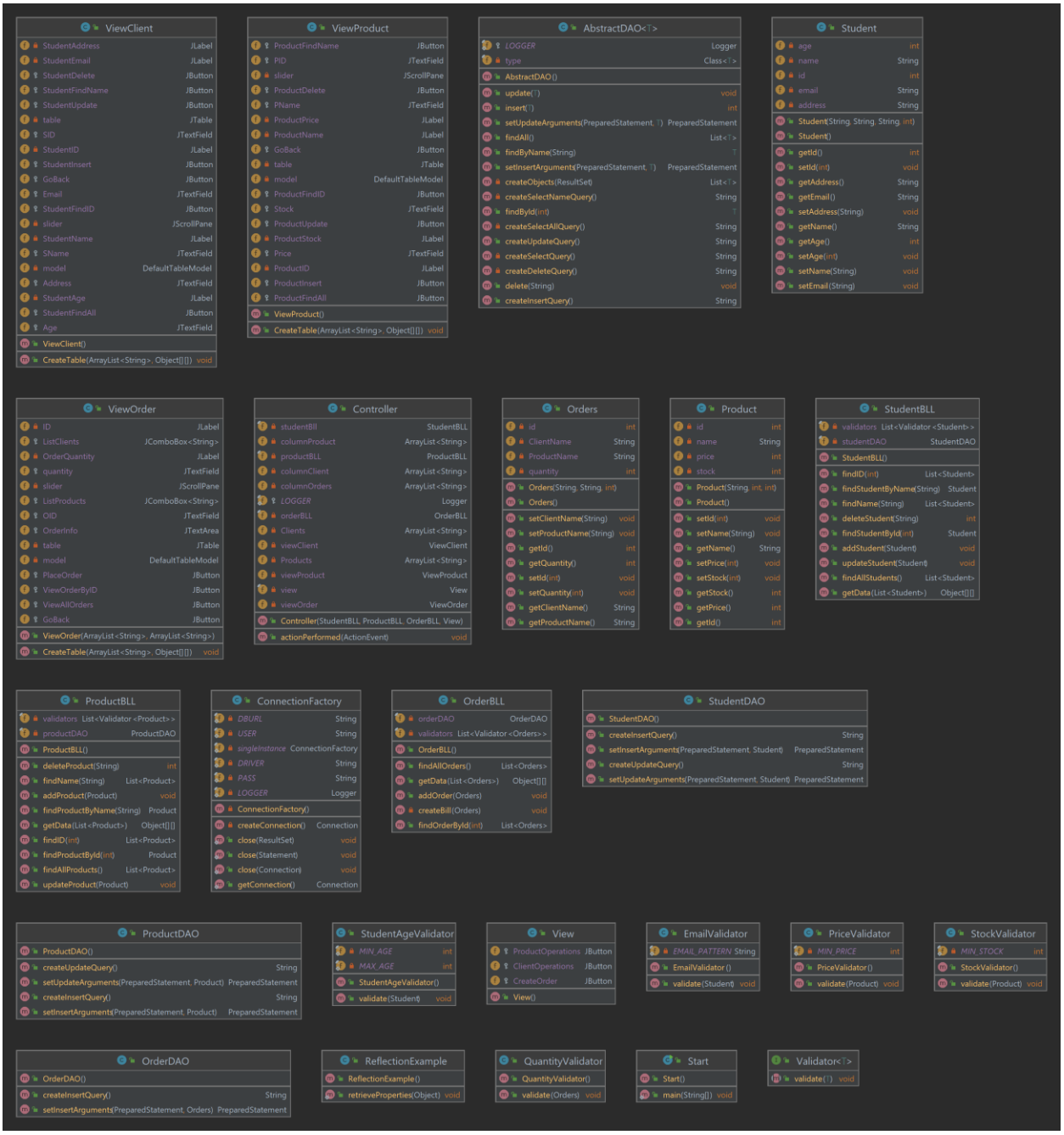
Diagrama UML de clase:

Ca și componente, aplicația este alcătuită din:

1. Clasele:
    - OrderBLL
    - ProductBLL
    - StudentBLL
    - EmailValidator
    - PriceValidator
    - QuantityValidator
    - StockValidator
    - StudentAgeValidator
    - ConnectionFactory
    - AbstractDAO<T>

- OrderDAO
- ProductDAO
- StudentDAO
- Orders
- Product
- Student
- Controller
- View
- ViewClient
- ViewOrder
- ViewProduct
- ReflectionExample
- Start

2. Interfața:
    - Validator<T>


ViewClient
StudentAddress JLabel
StudentEmail JLabel
StudentDelete JButton
StudentFindName JButton
StudentUpdate JButton
table JTable
SID JTextField
StudentID JLabel
StudentInsert JButton
GoBack JButton
Email JTextField
StudentFindID JButton
slider JScrollPane
StudentName JLabel
SName JTextField
model DefaultTableModel
Address JTextField
StudentAge JLabel
StudentFindAll JButton
Age JTextField
ViewClient()
CreateTable(ArrayList<String>, Object[][]) void
ViewProduct
ProductFindName JButton
PID JTextField
slider JScrollPane
ProductDelete JButton
PName JTextField
ProductPrice JLabel
ProductName JLabel
GoBack JButton
table JTable
model DefaultTableModel
ProductFindID JButton
Stock JTextField
ProductUpdate JButton
ProductStock JLabel
Price JTextField
ProductID JLabel
ProductInsert JButton
ProductFindAll JButton
ViewProduct()
CreateTable(ArrayList<String>, Object[][]) void
AbstractDAO<T>
LOGGER Logger
type Class<T>
AbstractDAO()
update(T) void
insert(T) int
setUpdateArguments(PreparedStatement, T) PreparedStatement
findAll() List<T>
findByName(String) T
setInsertArguments(PreparedStatement, T) PreparedStatement
createObjects(ResultSet) List<T>
createSelectNameQuery() String
findById(int) T
createSelectAllQuery() String
createUpdateQuery() String
createSelectQuery() String
createDeleteQuery() String
delete(String) void
createInsertQuery() String
Student
age int
name String
id int
email String
address String
Student(String, String, String, int)
Student()
getId() int
setId(int) void
getAddress() String
getEmail() String
setAddress(String) void
getName() String
getAge() int
setAge(int) void
setName(String) void
setEmail(String) void
ViewOrder
ID JLabel
ListClients JComboBox<String>
OrderQuantity JLabel
quantity JTextField
slider JScrollPane
ListProducts JComboBox<String>
OID JTextField
OrderInfo JTextArea
table JTable
model DefaultTableModel
PlaceOrder JButton
ViewOrderByID JButton
ViewAllOrders JButton
GoBack JButton
ViewOrder(ArrayList<String>, ArrayList<String>)
CreateTable(ArrayList<String>, Object[][]) void
Controller
studentBll StudentBLL
columnProduct ArrayList<String>
productBLL ProductBLL
columnClient ArrayList<String>
columnOrders ArrayList<String>
LOGGER Logger
orderBLL OrderBLL
Clients ArrayList<String>
viewClient ViewClient
Products ArrayList<String>
viewProduct ViewProduct
view View
viewOrder ViewOrder
Controller(StudentBLL, ProductBLL, OrderBLL, View)
actionPerformed(ActionEvent) void
Orders
id int
ClientName String
ProductName String
quantity int
Orders(String, String, int)
Orders()
setClientName(String) void
setProductName(String) void
getId() int
getQuantity() int
setId(int) void
setQuantity(int) void
getClientName() String
getProductName() String
Product
id int
name String
price int
stock int
Product(String, int, int)
Product()
setId(int) void
setName(String) void
getName() String
setPrice(int) void
setStock(int) void
getStock() int
getPrice() int
getId() int
StudentBLL
validators List<Validator<Student>>
studentDAO StudentDAO
StudentBLL()
findID(int) List<Student>
findStudentByName(String) Student
findName(String) List<Student>
deleteStudent(String) int
findStudentById(int) Student
addStudent(Student) void
updateStudent(Student) void
findAllStudents() List<Student>
getData(List<Student>) Object[][]
ProductBLL
validators List<Validator<Product>>
productDAO ProductDAO
ProductBLL()
deleteProduct(String) int
findName(String) List<Product>
addProduct(Product) void
findProductByName(String) Product
getData(List<Product>) Object[][]
findID(int) List<Product>
findProductById(int) Product
findAllProducts() List<Product>
updateProduct(Product) void
ConnectionFactory
DBURL String
USER String
singleInstance ConnectionFactory
DRIVER String
PASS String
LOGGER Logger
ConnectionFactory()
createConnection() Connection
close(ResultSet) void
close(Statement) void
close(Connection) void
getConnection() Connection
OrderBLL
orderDAO OrderDAO
validators List<Validator<Orders>>
OrderBLL()
findAllOrders() List<Orders>
getData(List<Orders>) Object[][]
addOrder(Orders) void
createBill(Orders) void
findOrderById(int) List<Orders>
StudentDAO
StudentDAO()
createInsertQuery() String
setInsertArguments(PreparedStatement, Student) PreparedStatement
createUpdateQuery() String
setUpdateArguments(PreparedStatement, Student) PreparedStatement
ProductDAO
ProductDAO()
createUpdateQuery() String
setUpdateArguments(PreparedStatement, Product) PreparedStatement
createInsertQuery() String
setInsertArguments(PreparedStatement, Product) PreparedStatement
StudentAgeValidator
MIN_AGE int
MAX_AGE int
StudentAgeValidator()
validate(Student) void
View
ProductOperations JButton
ClientOperations JButton
CreateOrder JButton
View()
EmailValidator
EMAIL_PATTERN String
EmailValidator()
validate(Student) void
PriceValidator
MIN_PRICE int
PriceValidator()
validate(Product) void
StockValidator
MIN_STOCK int
StockValidator()
validate(Product) void
OrderDAO
OrderDAO()
createInsertQuery() String
setInsertArguments(PreparedStatement, Orders) PreparedStatement
ReflectionExample
ReflectionExample()
retrieveProperties(Object) void
QuantityValidator
QuantityValidator()
validate(Orders) void
Start
Start()
main(String[]) void
Validator<T>
validate(T) void

4. Implementare

Se va descrie fiecare clasă cu câmpuri și metodele importante. Se va descrie implementarea interfeței utilizator.

Clasele StudentBLL, ProductBLL și OrderBLL inițializează validatorii proprii pentru verificarea datelor introduse de utilizator și prezintă ce operații asupra bazei de date se vor face.

Clasa de Data Access AbstractDAO creează metodele generice de acces a bazei de date, metode care vor fi accesate pentru fiecare tabel. În cadrul lor, prin intermediul procedeului reflexiei, sunt accesate și inițializate interogările tabelelor cu câmpurile aferente tabelului pentru care s-a apelat metoda.

Clasele Student, Product și Orders cuprind informațiile unui obiect de tipul respectiv, obiect care va reprezenta un rând al unui tabel. Numele claselor corespund cu numele tabelelor pentru a asigura legătura cu baza de date și a facilita accesul către proprietăți.

Interfața grafică este structurată după cum urmează:

1. Aplicația începe cu un meniu principal.

2. Utilizatorul este rugat să aleagă asupra cărui tabel dorește să facă interogări prin apăsarea unuia dintre cele trei butoane: "Client Operations", "Product Operations" și "Create Order".

3. Ajuns la tabelul dorit, va insera informațiile specifice tabelului și va alege ce operație dorește să execute.

4. Când consideră că a terminat cu interogările asupra tabelului curent, din cadrul oricărui tabel se poate întoarce la meniul principal prin apăsarea butonului "GO BACK".

5. Rezultate

Se vor prezenta scenariile pentru testare cu Junit sau alt Framework de testare.

Pentru testarea aplicației nu a fost necesară folosirea Junit sau a unui Framework de testare. Corectitudinea a fost demonstrată conectând programul la o bază de date MySQL și verificând rezultatele interogărilor.

Se compară conținutul tabelelor din cadrul aplicației MySQL server cu rezultatele afișate în interfața grafică, observându-se că se ajunge la rezultatele dorite.

Se poate observa și construcția și afișarea corectă a conținutului tabelelor în cadrul interfeței grafice.

6. Concluzii

Se vor prezenta concluziile, ce s-a învățat din temă, dezvoltări ulterioare.

Se poate remarca că aplicația de gestiune a comenzilor are comportamentul dorit. Pe lângă posibilitatea de a introduce prin intermediul aplicației înregistrări noi pentru tabelele Student, Product și Orders, rezultatele pot fi de asemenea vizualizate în cadrul aplicației.

Ca dezvoltări ulterioare se poate include îmbunătățirea experienței cu utilizatorul prin stilizarea mai în detaliu a interfeței grafice. De asemenea, aplicația ar putea beneficia de suport pentru a executa mai multe tipuri de interogări asupra datelor din tabel.

În cadrul acestei teme am învățat cum să utilizez clasele generice, metoda reflexiei și cum să conectez o bază de date la un program.

7. Bibliografie

https://www.baeldung.com/javadoc

https://stackhowto.com/how-to-add-row-dynamically-in-jtable-java/

https://www.tutorialspoint.com/how-to-add-a-new-row-to-jtable-with-insertrow-in-java-swing

https://dsrl.eu/courses/pt/materials/A3_Support_Presentation.pdf

https://www.baeldung.com/java-pdf-creation