

TEHNICI DE PROGRAMARE FUNDAMENTALE

**ORDER MANAGEMENT**

*Documentație*

Student

Nume: Souca Tania-Carina

Grupa:30223

Conținut:

1.Obiectivul temei

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

3.Proiectare

4.Implementare

5.Rezultate

6.Concluzii

7.Bibliografie

Capitolul I. *Obiectivul temei*

Obiectivul temei “Order Management” îl reprezintă proiectarea și implementarea unei aplicații care lucrează cu o bază de date și poate realiza management-ul unui firme (clienți, produse, comenzi).

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

Aplicația implementată va avea 3 tabele și anume: Client, Product, Orders, iar utilizatorul poate efectua operații de afișare, găsire, adăugare, ștergere, a elementelor din tabelele Client si Product și poate de asemenea să emită comenzi, selectând un client, un produs și cantitatea dorită(daca aceasta exista in stoc).

1. Rularea aplicației pe tabelul Client

Use case: Vom alege ce operatie se doreste sa se efectueze asupra tabelului client.Asadar un client poate fi adaugat in acest tabel,dar si sters.

Actor Principal: Client

Scenariul Principal:

* Clientul alege operația dorită
* Sistemul verifică dacă datele introduse sunt corecte
* În caz afirmativ, se vor observa modificarile in tabelul Client din MySQL.
* Tabelul se va actualiza după fiecare modificare făcută

1. Rularea aplicației pe tabelul Product

Use case: Vom alege ce operatie se doreste sa se efectueze asupra tabelului products.Asadar un produs poate fi adaugat in acest tabel,dar si sters.

Actor Principal: Client

Scenariul Principal:

* Clientul alege operația dorită
* Sistemul verifică dacă datele introduse sunt corecte
* În caz afirmativ, se vor observa modificarile in tabelul Product din MySQL.
* Tabelul se va actualiza după fiecare modificare făcută

1. Rularea aplicației pentru generarea comenzilor

Use case: Vom cauta in baza de date daca exista clientul care doreste sa lanseze comanda,in caz ca acesta exista va fii cautat in baza de date produsul dorit,in cazul in care si acesta exista va fii verificata cantitatea dorita.

Primary actor: Client

Scenariu Principal:

* Clientul specifica datele legate de numele produsului si cantitatea dorita.
* Sistemul verifică dacă datele introduse sunt corecte,daca acestea exista in baza de date

Capitolul III. *Proiectare*

Aplicația este structurată în pachete, cele principale fiind pachetele Bll, BllValidator, Connection, DAO, Model si Presentation în care sunt implementate structurile de date în care stocăm informația despre tabelele din BD, generearea și repartizarea informațiilor în simulare .

**Diagrama UML :**

A screenshot of a computer

Description automatically generated with medium confidence

Capitolul IV.  *Implementare*

Aplicația este strucutrată ierarhic, pe pachecte în care fiecare layer are un scop special.

Aplicația este structurată în următoarele Layer: Model – modelează tabelele din baza de date, Data Access layer – conține clasele cu interogări și conexiunea la BD, Business Layer – conține clasele care încapsulează logica aplicației.

Pentru a ușura întelegerea implementării aplicației, fiecare funcție va avea un javadoc în care este explicată utilizarea funcției.

Pachetul Model conține clasele Client, Product si Orders și acestea sunt mapate direct pe tabelele din baza de date.

În pachetul Connection, avem implementată clasa ConnectionFactory care realizează conexiunea cu BD:

public class ConnectionFactory {  
 private static final Logger LOGGER = Logger.getLogger(ConnectionFactory.class.getName());  
 private static final String DRIVER = "com.mysql.jdbc.Driver";  
 private static final String DBURL = "jdbc:mysql://localhost:3306/tema3tp";  
 private static final String USER = "TaniaSouca";  
 private static final String PASS = "pass";  
 private static ConnectionFactory singleInstance = new ConnectionFactory();  
 private ConnectionFactory() {  
 private Connection createConnection() {  
 Connection connection = null;  
 try {  
 connection = DriverManager.getConnection(DBURL, USER, PASS);  
 } catch (SQLException e) {  
 LOGGER.log(Level.WARNING, "An error occured while trying to connect to the database");  
 e.printStackTrace();  
 }  
 return connection;  
 }  
 public static void close(Connection connection) {  
 if (connection != null) {  
 try {  
 connection.close();  
 } catch (SQLException e) {  
 LOGGER.log(Level.WARNING, "An error occured while trying to close the connection");  
 }  
 }  
 }  
 public static void close(Statement statement) {  
 if (statement != null) {  
 try {  
 statement.close();  
 } catch (SQLException e) {  
 LOGGER.log(Level.WARNING, "An error occured while trying to close the statement");  
 }  
 }  
 }  
 public static void close(ResultSet resultSet) {  
 if (resultSet != null) {  
 try {  
 resultSet.close();  
 } catch (SQLException e) {  
 LOGGER.log(Level.WARNING, "An error occured while trying to close the ResultSet");  
 }  
 }  
 }  
 public static Connection getConnection() {  
 return singleInstance.createConnection();  
 }  
}

În pachetul Bll.Validator avem definite clase care implementează interfața Validator pentru verificarea datelor introduse.

Un exemplu ar fii cautarea cantitatii produsului dorit:

public class OrderQuantityValidator implements Validator<Orders> {  
 @Override  
 public void validate(Orders t) {  
 if (t.getQuantity()<0) {  
 throw new IllegalArgumentException("Error:negative quantity");  
 } else {  
 ProductBll productBLL = new ProductBll();  
 Product product = productBLL.findProductById(t.getProductID());  
 if(product==null) {  
 throw new IllegalArgumentException("Product with id = " + t.getProductID() + " was not found!");  
 } else {  
 if(t.getQuantity()>product.getInitQuantity())  
 throw new IllegalArgumentException("Error:doesn't exist this quantity");  
 else  
 System.out.println("Order taken!");  
 }  
 }  
 }

Clasele din pachetul BLL reprezintă clasele de business pentru aplicația noastră. De exemplu, pentru tabelul Product, avem:

/\*\*  
 \* Clasa Business --> pentru produse  
 \* @author tania  
 \*/  
  
public class ProductBll {  
 private List<Validator<Product>> validators;  
 private ProductDAO productDAO;  
  
 public ProductBll() {  
 validators = new ArrayList<Validator<Product>>();  
 validators.add(new ProductPriceValidator());  
 validators.add(new ProductQuantityValidator());  
 productDAO = new ProductDAO();  
 }  
  
 public List<Product> findAllProducts() {  
 List<Product> list = productDAO.findAll();  
 if(list==null) {  
 throw new NoSuchElementException("Error:Doesn't exist products in db");  
 }  
 return list;  
 }  
  
 public Product findProductByName(String name){  
 Product prod = productDAO.findByName(name);  
 if(prod == null)  
 throw new NoSuchElementException("The product named '"+name+"' doesn't exist!");  
 return prod;  
 }  
  
 public Product findProductById(int id) {  
 Product prod = productDAO.findById(id);  
 if (prod == null) {  
 throw new NoSuchElementException("The product with id =" + id + " was not found!");  
 }  
 return prod;  
 }

În pachetul DAO, clasa AbstractDAO este o clasă care efectueaza operațiile de find, insert, update, delete pentru orice obiect care se dă ca parametru.

public T findById(int id) {  
 Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 String query = createSelectQuery("id");  
 try {  
 try {  
 Class.forName("com.mysql.jdbc.Driver");  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 connection = ConnectionFactory.getConnection();  
 statement = connection.prepareStatement(query);  
 statement.setInt(1, id);  
 resultSet = statement.executeQuery();  
 return createObjects(resultSet).get(0);  
 } catch (SQLException e) {  
 LOGGER.log(Level.WARNING, type.getName() + "DAO:findById " + e.getMessage());  
 } finally {  
 ConnectionFactory.close(resultSet);  
 ConnectionFactory.close(statement);  
 ConnectionFactory.close(connection);  
 }  
 return null;  
}

Capitolul V. *Rezultate*

În acest capitol, vom prezenta rezultatele admise in urma folosirii aplicatiei „**ORDER MANAGEMENT**”.

-Insert client:clientul propus pentru inserare impreuna cu numele si orasul sau vor fii introduse in tabela client din baza de date.

-Delete client:clientul propus pentru stergere impreuna cu numele si orasul sau vor fii sterse din tabela client din baza de date.

-Insert product:produsul propus pentru inserare impreuna cu numele , cantitatea si pretul sau vor fii introduse in tabela product din baza de date.

-Delete product:produsul propus pentru stergere impreuna cu numele,cantitatea si pretul sau vor fii sterse din tabela product din baza de date.

Order: aceasta va lansa o comanda in care se gaseste numele clientului,numele produsului dorit ,precum si cantitatea dorita.

Capitolul VI. *Concluzii*

Această temă este importantă deoarece utilizează concepte din teoria bazelor de date și implementarea aplicațiilor cu baze de date intergrate prin Java. Totodată, această problemă stă la baza modelării proceselor din viața reală și rezolvarea acesteia oferă un avantaj atât în evidențierea trăsăturilor programării cu baze de date, cât și în abstractizarea cât mai eficientă a problemelor din viața reală pentru implementare software.

Capitolul VI. *Bibliografie*

<https://www.geeksforgeeks.org/java/>