**ORDERS MANAGEMENT**

Tema 3

DOCUMENTAȚIE

**Gavrilescu Andreea-Lavinia**

**Grupa 30228**

# CUPRINS

[1. **Obiectivul temei**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297885) 1

[2. **Analiza problemei, modelare, scenarii, cazuri de utilizare**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297886) 2

[3. **Proiectare**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297887) 8

[4. **Implementare**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297888) 13

[5. **Rezultate**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297889) 31

[6. **Concluzii**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297890) 32

[7. **Bibliografie**](file:///D:\Facultate\UTCN\An%20II\Semestrul%202\Tehnici%20de%20programare\Laboratoare\PT2021-2022_Documentation_Template.doc#_Toc95297891) 33

Obiectivul temei

**Obiective principale**

* Proiectarea și implementarea unei aplicații care are ca scop administrarea unui depozit.

**Obiective secundare**

* Analizarea problemei și identificarea cerințelor
* Proiectarea aplicației de administrare a depozitului
* Implementarea aplicației de administrare a depozitului
* Testarea aplicației de administrare a depozitului

Analiza problemei, modelare, scenarii, cazuri de utilizare

**Analiza**

ORDERS MANAGEMENT

**Cerințe funcționale**

* Aplicația de administrare ar trebui să permită angajatului să introducă un nou client
* Aplicația de administrare ar trebui să permită angajatului să introducă un nou produs
* Aplicația de administrare ar trebui să permită angajatului să introducă o nouă comandă
* Aplicația de administrare ar trebui să permită angajatului să modifice un nou client
* Aplicația de administrare ar trebui să permită angajatului să modifice un nou produs
* Aplicația de administrare ar trebui să permită angajatului să șteargă un nou client
* Aplicația de administrare ar trebui să permită angajatului să șteargă un nou produs

**Cerințe non-funcționale**

* Aplicația de simulare ar trebui să fie interactivă
* Aplicația de simulare ar trebui să fie ușor de folosit de către utilizator

**Cazuri de utilizare**

* Caz de utilizare:adăugare client

Actor principal: angajatul

Scenariu principal

1. Angajatul completează câmpurile necesare cu informațiile clientului
2. Angajatul apasă pe butonul de “ADAUGARE”
3. Aplicația introduce clientul în baza de date și afișează un mesaj corespunzător

Scenariu alternativ

1. Angajatul nu introduce date în toate câmpurile necesare
2. Aplicația un mesaj de eroare corespunzător
3. Scenariul se întoarce la punctul 1

* Caz de utilizare:adăugare produs

Actor principal: angajatul

Scenariu principal

1. Angajatul completează câmpurile necesare cu informațiile produsului

2. Angajatul apasă pe butonul de “ADAUGARE”

3. Aplicația introduce produsul în baza de date și afișează un mesaj corespunzător

Scenariu alternativ

1. Angajatul nu introduce date în toate câmpurile necesare

2. Aplicația un mesaj de eroare corespunzător

3. Scenariul se întoarce la punctul 1

* Caz de utilizare:adăugare comandă

Actor principal: angajatul

Scenariu principal

1. Angajatul completează câmpul pentru ID, cantitate și selectează un client și un produs din tabelele corespunzătoare

2. Angajatul apasă pe butonul de “ADAUGARE”

3. Aplicația introduce comanda în baza de date și afișează un mesaj corespunzător

Scenariu alternativ

1. Angajatul nu introduce date valide în toate câmpul necesar

2. Aplicația un mesaj de eroare corespunzător

3. Scenariul se întoarce la punctul 1

* Caz de utilizare:ștergere client

Actor principal: angajatul

Scenariu principal

1. Angajatul completează câmpurile necesare cu informațiile clientului (se poate completa doar ID-ul clientului)

2. Angajatul apasă pe butonul de “ȘTERGERE”

3. Aplicația șterge clientul din baza de date și afișează un mesaj corespunzător

Scenariu alternativ

1. Angajatul nu introduce date în toate câmpurile necesare

2. Aplicația un mesaj de eroare corespunzător

3. Scenariul se întoarce la punctul 1

* Caz de utilizare:ștergere produs

Actor principal: angajatul

Scenariu principal

1. Angajatul completează câmpurile necesare cu informațiile produsului (se poate completa doar ID-ul produsului)

2. Angajatul apasă pe butonul de “ȘTERGERE”

3. Aplicația șterge produsul din baza de date și afișează un mesaj corespunzător

Scenariu alternativ

1. Angajatul nu introduce date în toate câmpurile necesare

2. Aplicația un mesaj de eroare corespunzător

3. Scenariul se întoarce la punctul 1

* Caz de utilizare:modificare client

Actor principal: angajatul

Scenariu principal

1. Angajatul completează câmpurile necesare cu informațiile clientului

2. Angajatul apasă pe butonul de “MODIFICARE”

3. Aplicația modifică clientul din baza de date și afișează un mesaj corespunzător

Scenariu alternativ

1. Angajatul nu introduce date în toate câmpurile necesare

2. Aplicația un mesaj de eroare corespunzător

3. Scenariul se întoarce la punctul 1

* Caz de utilizare:modificare produs

Actor principal: angajatul

Scenariu principal

1. Angajatul completează câmpurile necesare cu informațiile produsului

2. Angajatul apasă pe butonul de “MODIFICARE”

3. Aplicația modifică produsul din baza de date și afișează un mesaj corespunzător

Scenariu alternativ

1. Angajatul nu introduce date în toate câmpurile necesare

2. Aplicația un mesaj de eroare corespunzător

3. Scenariul se întoarce la punctul 1

* Caz de utilizare:afișare client

Actor principal: angajatul

Scenariu principal

1. Angajatul apasă pe butonul de “AFISARE”

2. Aplicația afișează toți clienții care au fost adăugați în baza de date

Scenariu alternativ

1. Scenariul se întoarce la punctul 1

* Caz de utilizare:afișare produs

Actor principal: angajatul

Scenariu principal

1. Angajatul apasă pe butonul de “AFISARE”

2. Aplicația afișează toate produsule care au fost adăugate în baza de date

Scenariu alternativ

1. Scenariul se întoarce la punctul 1

* Caz de utilizare:afișare comadă

Actor principal: angajatul

Scenariu principal

1. Angajatul apasă pe butonul de “AFISARE”

2. Aplicația afișează toate comenzile care au fost adăugate în baza de date

Scenariu alternativ

1. Scenariul se întoarce la punctul 1

Proiectare

**Design general**

Client

Produs

Comandă

Date

**Divizare în pachete**

Conține clasele care implementează interfața

Presentation Layer

Business Logic

Conține clasele care implementează operațiile

Model

Conține clasele care conțin query-urile și conexiunea cu baza de date

Data Layer

Conține clasele care modelează datele aplicației relaționate cu baza de date

Database

Pentru afișarea rezulatelor potrivite confrom datelor introduse de utilizatori prin interfața grafică voi folosi un Model-View-Controller pattern care împarte aplicația în 3 categorii: procesare, input și output.

* Componentele Model – încapsulează datele și funcționalitatea
* Componentele View – afișează informația utilizatorului – obține datele afișate din model
* Controller – fiecare view are un controller asociat; acesta primește un input, de obicei un eveniment care denotă mișcarea mouse-ului, apăsarea mouse-ului sau a tastaturii; aceste evenimente sunt traduse ca apeluri care sunt trimise către model sau view.

Controller-ul modifică apeluri View-ului în funcție de modificările din Model

Datele din Model sunt folosite în View

Model

View-ul poate cere date din Model

View

Controller

Interacțiunea cu View-ul face apeluri către Controller

**Divizare în clase**

Client

Model

Product

Order

ClientDAO

DAO

ProductDAO

OrderDAO

ProductBLL

ClientBLL

OrderBLL

BLL

ClientController

Controller

ProductController

OrderController

ClientView

View

ProductView

OrderView

ConnectionFactory

Connection

Start

Main

OrderFile

O imagine care conține text

Descriere generată automat

Pachetul Models - conține 3 clase pentru a asigura lizibilitatea codului, dar și intuitivitatea, astfel:

* Clasa Client – conține 3 atribute numite sugestiv, ID,nume și adresa de mail, precum și un constructor, metodele de get() și set().
* Clasa Product – conține 3 atribute numite sugestiv, ID,nume, preț, cantitate, precum și un constructor, metodele de get() și set().
* Clasa Order – conține 3 atribute numite sugestiv, ID, IDClient, IDProdus și cantitate, precum și un constructor, metodele de get() și set().

Pachetul dao - conține 4 clase pentru a asigura lizibilitatea codului, dar și intuitivitatea, astfel:

* Clasa AbstractDAO – are ca atribut un type generic, un constructor, precum și metode precum crearea generică a unor obiecte, crearea query-urilor necesare, precum și metodele de afișare după un ID căutat și a tuturor clienților, insert, update și delete a unor obiecte în mod general.
* Clasa ClientDAO – extinde clasa AbstractDAO.
* Clasa ProductDAO – extinde clasa AbstractDAO.
* Clasa OrderDAO – extinde clasa AbstractDAO.

Pachetul bll - conține 3 clase și un pachet pentru a asigura lizibilitatea codului, dar și intuitivitatea, astfel:

* Pachetul validators - conține 3 clase și un pachet pentru a asigura lizibilitatea codului, dar și intuitivitatea, astfel:
* Clasa ClientValidator– verifică datele pentru client.
* Clasa ProductValidator – verifică datele pentru produs.
* Clasa OrderValidator – verifică datele pentru comandă.
* Interfața Validator – extinsă de clasele de mai sus.
* Clasa ClientBLL – implementează logica pentru clasa ClientDAO.
* Clasa ProductBLL – implementează logica pentru clasa ProductDAO.
* Clasa OrderBLL– implementează logica pentru clasa OrderDAO.

Pachetul View – conține clasele ClientView, ProductView și OrderView care implementează interfețele folosind Java Swing.

Pachetul Controller – conține clasa ClientController, ProductController și OrderController care receptează acțiunile angajatului și returnează rezultatul potrivit prin interfața grafică.

Pachetul Start – conține clasele Main și OrderFile.

* Clasa Main – care are ca atribute, view-urile și controller-ele, validatori și dao-urile.
* Clasa OrderFile – care are o metodă care scrie într-un fișier.

Pachetul Connection – conține clasa ConnectionFactory care face legătura cu baza de date.

Implementare

**1. Clasa Client** – are ca atribute private, ID-ul de tipul int, numele și adresa de tipul String, însoțite de metodele standard de get() și set().

**2. Clasa Product** – are ca atribute private, ID-ul și cantitatea de tipul int, numele de tipul String, prețul de tip double, însoțite de metodele standard de get() și set().

**3.** **Clasa Order** – are ca atribute private, ID-ul, IDClient, IDProdus și cantitatea de tipul int, însoțite de metodele standard de get() și set().

**4.** **Clasa AbstractDAO** – are ca atribut private, type-ul de tipul Class<T>, precum și un constructor care ia obiectele din clasa corespunzătoare utilizând reflexia.

*/\*\*  
 \* Constructorul clasei AbstractDAO  
 \*/*@SuppressWarnings*(*"unchecked"*)*public AbstractDAO*() {* this.type = *(*Class*<*T*>) ((*ParameterizedType*)* getClass*()*.getGenericSuperclass*())*.getActualTypeArguments*()[*0*]*;  
  
*}*

Cât despre metodele care folosesc reflexia, avem:

*/\*\*  
 \* Metoda care creeaza o lista de obiecte de tipul corespunzator tabelelor respective  
 \*  
 \** ***@param*** *resultSet contine field-urile fiecarei tabele  
 \** ***@return*** *o lista de obiecte de tipul corespunzator tabelei corespunzatoare  
 \*/*private List*<*T*>* createObjects*(*ResultSet resultSet*) {* List*<*T*>* list = new ArrayList*<*T*>()*;  
 Constructor*[]* ctors = type.getDeclaredConstructors*()*;  
 Constructor ctor = null;  
 for *(*int i = 0; i < ctors.length; i++*) {* ctor = ctors*[*i*]*;  
 if *(*ctor.getGenericParameterTypes*()*.length == 0*)* break;  
 *}* try *{* while *(*resultSet.next*()) {* ctor.setAccessible*(*true*)*;  
 T instance = *(*T*)* ctor.newInstance*()*;  
 for *(*Field field : type.getDeclaredFields*()) {* String fieldName = field.getName*()*;  
 Object value = resultSet.getObject*(*fieldName*)*;  
 PropertyDescriptor propertyDescriptor = new PropertyDescriptor*(*fieldName, type*)*;  
 Method method = propertyDescriptor.getWriteMethod*()*;  
 method.invoke*(*instance, value*)*;  
 *}* list.add*(*instance*)*;  
 *}  
 }* catch *(*InstantiationException e*) {* e.printStackTrace*()*;  
 *}* catch *(*IllegalAccessException e*) {* e.printStackTrace*()*;  
 *}* catch *(*SecurityException e*) {* e.printStackTrace*()*;  
 *}* catch *(*IllegalArgumentException e*) {* e.printStackTrace*()*;  
 *}* catch *(*InvocationTargetException e*) {* e.printStackTrace*()*;  
 *}* catch *(*SQLException e*) {* e.printStackTrace*()*;  
 *}* catch *(*IntrospectionException e*) {* e.printStackTrace*()*;  
 *}* return list;  
*}  
  
/\*\*  
 \* Metoda care construieste query-ul pentru SELECT  
 \*  
 \** ***@param*** *field contine un field  
 \** ***@return*** *query-ul obtinut  
 \*/*private String createSelectQuery*(*String field*) {* StringBuilder sb = new StringBuilder*()*;  
 sb.append*(*"SELECT "*)*;  
 sb.append*(*" \* "*)*;  
 sb.append*(*" FROM `"*)*;  
 sb.append*(*type.getSimpleName*())*;  
 sb.append*(*"` WHERE " + field + " =?"*)*;  
 return sb.toString*()*;  
*}  
  
/\*\*  
 \* Metoda care construieste query-ul pentru INSERT  
 \*  
 \** ***@return*** *query-ul obtinut  
 \*/*private String createInsertQuery*() {* String str = "INSERT INTO `" + type.getSimpleName*()* + "`";  
 str += " VALUES (";  
  
 int nrAtribute = type.getDeclaredFields*()*.length;  
 for *(*int i = 0; i <= nrAtribute - 2; i++*)* str += "?, ";  
  
 str += "?)";  
  
 return str;  
*}  
  
/\*\*  
 \* Metoda care construieste query-ul pentru UPDATE  
 \*  
 \** ***@return*** *query-ul obtinut  
 \*/*private String createUpdateQuery*() {* String str = "UPDATE `" + type.getSimpleName*()* + "` SET ";  
  
 Field*[]* fields = type.getDeclaredFields*()*;  
 int nrAtribute = fields.length;  
  
 for *(*int i = 0; i <= nrAtribute - 3; i++*) {* str += fields*[*i + 1*]*.getName*()* + " = ?, ";  
 *}* str += fields*[*nrAtribute - 1*]*.getName*()* + " = ? WHERE id = ?";  
  
 return str;  
*}  
  
/\*\*  
 \* Metoda care construieste query-ul pentru DELETE  
 \*  
 \** ***@return*** *query-ul obtinut  
 \*/*private String createDeleteQuery*() {* String str = "DELETE FROM `" + type.getSimpleName*()* + "` WHERE id = ?";  
  
 return str;  
*}  
  
/\*\*  
 \* Metoda care afiseaza toate datele dintr-o tabela  
 \*  
 \** ***@return*** *o lista de obiecte de tipul corespunzator tabelei respective  
 \*/*public List*<*T*>* findAll*() {* Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 String query = "SELECT \* FROM " + type.getSimpleName*()*;  
  
 try *{* connection = ConnectionFactory.*getConnection()*;  
 statement = connection.prepareStatement*(*query*)*;  
 resultSet = statement.executeQuery*()*;  
  
 return createObjects*(*resultSet*)*;  
  
 *}* catch *(*SQLException e*) {* e.printStackTrace*()*;  
 *}* finally *{* ConnectionFactory.*close(*resultSet*)*;  
 ConnectionFactory.*close(*statement*)*;  
 ConnectionFactory.*close(*connection*)*;  
 *}* return null;  
*}  
  
/\*\*  
 \* Metoda care afiseaza toate datele dintr-o tabela dupa parametrul dat  
 \*  
 \** ***@param*** *id ID-ul dupa care se cauta datele  
 \** ***@return*** *rezultatul corespunzator  
 \*/*public T findById*(*int id*) {* Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
 String query = createSelectQuery*(*"id"*)*;  
 try *{* connection = ConnectionFactory.*getConnection()*;  
 statement = connection.prepareStatement*(*query*)*;  
 statement.setInt*(*1, id*)*;  
 resultSet = statement.executeQuery*()*;  
  
 return createObjects*(*resultSet*)*.get*(*0*)*;  
 *}* catch *(*SQLException e*) {* e.printStackTrace*()*;  
 *}* finally *{* ConnectionFactory.*close(*resultSet*)*;  
 ConnectionFactory.*close(*statement*)*;  
 ConnectionFactory.*close(*connection*)*;  
 *}* return null;  
*}  
  
/\*\*  
 \* Metoda care insereaza date intr-o tabela  
 \*  
 \** ***@param*** *t obiectul care trebuie inserat  
 \** ***@return*** *obiectul inserat  
 \*/*public T insert*(*T t*) {* Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
  
 try *{* connection = ConnectionFactory.*getConnection()*;  
 statement = connection.prepareStatement*(*this.createInsertQuery*())*;  
  
 int index = 1;  
  
 for *(*Field f : type.getDeclaredFields*()) {* f.setAccessible*(*true*)*;  
  
 if *(*f.get*(*t*)* instanceof Integer*) {* statement.setInt*(*index, *(*Integer*)* f.get*(*t*))*;  
 *}* else if *(*f.get*(*t*)* instanceof String*) {* statement.setString*(*index, *(*String*)* f.get*(*t*))*;  
 *}* else if *(*f.get*(*t*)* instanceof Double*) {* statement.setDouble*(*index, *(*Double*)* f.get*(*t*))*;  
 *}* index++;  
 *}* statement.executeUpdate*()*;  
  
 return t;  
 *}* catch *(*SQLException | IllegalAccessException ex*) {* ex.printStackTrace*()*;  
 *}* finally *{* ConnectionFactory.*close(*resultSet*)*;  
 ConnectionFactory.*close(*statement*)*;  
 ConnectionFactory.*close(*connection*)*;  
 *}* return null;  
*}  
  
/\*\*  
 \* Metoda care modifica date dintr-o tabela  
 \*  
 \** ***@param*** *t obiectul care trebuie modificat  
 \** ***@return*** *obiectul modificat  
 \*/*public T update*(*T t*) {* Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
  
 try *{* connection = ConnectionFactory.*getConnection()*;  
 statement = connection.prepareStatement*(*this.createUpdateQuery*())*;  
  
 int index = 1;  
  
 Field*[]* fields = type.getDeclaredFields*()*;  
 for *(*int i = 1; i <= fields.length - 1; i++*) {* fields*[*i*]*.setAccessible*(*true*)*;  
  
 if *(*fields*[*i*]*.get*(*t*)* instanceof Integer*) {* statement.setInt*(*index, *(*Integer*)* fields*[*i*]*.get*(*t*))*;  
 *}* else if *(*fields*[*i*]*.get*(*t*)* instanceof String*) {* statement.setString*(*index, *(*String*)* fields*[*i*]*.get*(*t*))*;  
 *}* else if *(*fields*[*i*]*.get*(*t*)* instanceof Double*) {* statement.setDouble*(*index, *(*Double*)* fields*[*i*]*.get*(*t*))*;  
 *}* index++;  
 *}* fields*[*0*]*.setAccessible*(*true*)*;  
 statement.setInt*(*index, *(*Integer*)* fields*[*0*]*.get*(*t*))*;  
  
 statement.executeUpdate*()*;  
  
 return t;  
 *}* catch *(*SQLException | IllegalAccessException ex*) {* ex.printStackTrace*()*;  
 *}* finally *{* ConnectionFactory.*close(*resultSet*)*;  
 ConnectionFactory.*close(*statement*)*;  
 ConnectionFactory.*close(*connection*)*;  
 *}* return null;  
*}  
  
/\*\*  
 \* Metoda care sterge date dintr-o tabela  
 \*  
 \** ***@param*** *t obiectul care trebuie sters  
 \** ***@return*** *obiectul sters  
 \*/*public T delete*(*T t*) {* Connection connection = null;  
 PreparedStatement statement = null;  
 ResultSet resultSet = null;  
  
 try *{* connection = ConnectionFactory.*getConnection()*;  
 statement = connection.prepareStatement*(*this.createDeleteQuery*())*;  
  
 Field id = type.getDeclaredFields*()[*0*]*;  
 id.setAccessible*(*true*)*;  
  
 statement.setInt*(*1, *(*Integer*)* id.get*(*t*))*;  
  
 statement.executeUpdate*()*;  
  
 return t;  
 *}* catch *(*SQLException | IllegalAccessException ex*) {* ex.printStackTrace*()*;  
 *}* finally *{* ConnectionFactory.*close(*resultSet*)*;  
 ConnectionFactory.*close(*statement*)*;  
 ConnectionFactory.*close(*connection*)*;  
 *}* return null;  
*}*

**5.** **Clasa ClientDAO** – extinde clasa AbstractDAO.

**6.** **Clasa ProductDAO** – extinde clasa AbstractDAO.

**7.** **Clasa OrderDAO** – extinde clasa AbstractDAO.

**8.** **Clasa ClientValidator** – verifică prin metode datele pentru client.

*/\*\*  
 \* Metoda cade valideaza atributele specifice unei comenzi  
 \*  
 \** ***@param*** *order comanda pentru care se vor face validari  
 \** ***@throws*** *Exception daca datele comenzii sunt invalide  
 \*/*@Override  
public void validate*(*Order order*)* throws Exception *{* if *(*order.getCantitate*()* < 1*)* // verificare cantitate  
 throw new Exception*(*"Cantitate invalida pentru comanda!"*)*;  
*}*

**9.** **Clasa ProductValidator** – verifică prin metode datele pentru produs.

**10.** **Clasa OrderValidator** – verifică prin metode datele pentru comandă.

*/\*\*  
 \* Metoda cade valideaza atributele specifice unei comenzi  
 \*  
 \** ***@param*** *order comanda pentru care se vor face validari  
 \** ***@throws*** *Exception daca datele comenzii sunt invalide  
 \*/*@Override  
public void validate*(*Order order*)* throws Exception *{* if *(*order.getCantitate*()* < 1*)* // verificare cantitate  
 throw new Exception*(*"Cantitate invalida pentru comanda!"*)*;  
*}*

**11.** **Interfața Validator** – implementată de clasele de mai sus.

*/\*\*  
 \* Interfata folosita pentru clasele validatoare de obiecte  
 \*  
 \** ***@param*** <*T*> *tipul obiectului care urmeaza sa fie verificat  
 \*/*public interface Validator*<*T*> {* public void validate*(*T t*)* throws Exception;  
*}*

**12.** **Clasa ClientBLL** – are ca atribute de tip private un validator și un clientDAO, un constructor cu aceste atribute, precum și metode care conectează logica din spatele operațiilor din DAO cu baza de date.

*/\*\*  
 \* Constructor pentru clasa ClientBLL  
 \*  
 \** ***@param*** *validator validatorul pentru client  
 \** ***@param*** *clientDAO dao-ul pentru client responsabil pentru operatiile pe tabela Client  
 \*/*public ClientBLL*(*Validator*<*Client*>* validator, ClientDAO clientDAO*) {* this.validator = validator;  
 this.clientDAO = clientDAO;  
*}  
  
/\*\*  
 \* Metoda care returneaza toti clientii din tabela Client  
 \*  
 \** ***@return*** *lista de clienti din tabela Client  
 \*/*public List*<*Client*>* findAll*() {* return this.clientDAO.findAll*()*;  
*}  
  
/\*\*  
 \* Metoda care returneaza clientul cu ID-ul corespunzator  
 \*  
 \** ***@param*** *id ID-ul dupa care se cauta clientul in tabela Client  
 \** ***@return*** *clientul cu ID-ul cautat  
 \** ***@throws*** *Exception daca nu exista un client cu ID-ul dat ca parametru  
 \*/*public Client findById*(*int id*)* throws Exception *{* Client client = this.clientDAO.findById*(*id*)*;  
 if *(*client == null*)* throw new Exception*(*"Nu exista niciun client cu acest ID!"*)*;  
  
 return client;  
*}  
  
/\*\*  
 \* Metoda in care se adauga un client in tabela Client  
 \*  
 \** ***@param*** *client clientul care trebuie sa fie adaugat in tabela Client  
 \** ***@return*** *clientul care a fost adaugat  
 \** ***@throws*** *Exception daca exista deja un client cu acel ID  
 \*/*public Client insert*(*Client client*)* throws Exception *{* this.validator.validate*(*client*)*; // validare client  
  
 if *(*this.clientDAO.findById*(*client.getId*())* != null*)* throw new Exception*(*"Exista deja un client cu acest ID!"*)*;  
  
 return this.clientDAO.insert*(*client*)*;  
*}  
  
/\*\*  
 \* Metoda in care se sterge un client in tabela Client  
 \*  
 \** ***@param*** *client clientul care trebuie sa fie sters in tabela Client  
 \** ***@return*** *clientul trebuie sters  
 \** ***@throws*** *Exception daca nu exista clientul respectiv  
 \*/*public Client delete*(*Client client*)* throws Exception *{* if *(*this.clientDAO.findById*(*client.getId*())* == null*)* throw new Exception*(*"Nu exista acest client!"*)*;  
  
 Client rezultat = this.clientDAO.delete*(*client*)*; // stergere client  
  
 return rezultat;  
*}  
  
/\*\*  
 \* Metoda in care se modifica un client in tabela Client  
 \*  
 \** ***@param*** *client clientul care trebuie sa fie modificat in tabela Client  
 \** ***@return*** *clientul care a fost modificat  
 \** ***@throws*** *Exception daca clientul respectiv nu poate fi modificat  
 \*/*public Client update*(*Client client*)* throws Exception *{* Client rezultat = this.findById*(*client.getId*())*;  
  
 if *(*rezultat == null*)* throw new Exception*(*"Nu se poate face update la acest client!"*)*;  
  
 return this.clientDAO.update*(*client*)*;  
*}*

**13.** **Clasa ProductBLL** – are ca atribute de tip private un validator și un productDAO, un constructor cu aceste atribute, precum și metode care conectează logica din spatele operațiilor din DAO cu baza de date.

**14.** **Clasa ClientBLL** – are ca atribute de tip private un validator și un orderDAO, un constructor cu aceste atribute, precum și metode care conectează logica din spatele operațiilor din DAO cu baza de date.

**15.** **Clasa ClientView** – impementează elementele interfeței grafice pentru Client

– conține atributele necesare în casete text, a etichetelor corepunzătoare acestora, precum și a butoanelor de adăugare, modificare, ștergere și afișare.

În constructor vom poziționa toate elementele interfeței, pe lângă metodele de get() și set() pentru castele text, avem metode de adăugare a unor listeners pentru a verifica dacă unul dintre butoane este apăsat, caz în care se trimite un apel către controller pentru a afișa rezulatul corepunzător.

*/\*\*  
 \* Constructor pentr clasa ClientView  
 \*/*public ClientView*() {* // Marginile ferestrei interfetei  
 this.setBounds*(*100, 100, 1100, 550*)*;  
 this.setDefaultCloseOperation*(*JFrame.*EXIT\_ON\_CLOSE)*;  
 this.getContentPane*()*.setLayout*(*null*)*;  
  
 // Etichete  
 titleLabel = new JLabel*(*"CLIENT TABLE"*)*;  
 titleLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 18*))*;  
 titleLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 titleLabel.setBounds*(*578, 21, 171, 46*)*;  
 this.getContentPane*()*.add*(*titleLabel*)*;  
  
 idLabel = new JLabel*(*"ID:"*)*;  
 idLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 idLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 18*))*;  
 idLabel.setBounds*(*10, 77, 74, 36*)*;  
 this.getContentPane*()*.add*(*idLabel*)*;  
  
 numeLabel = new JLabel*(*"NUME:"*)*;  
 numeLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 numeLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 18*))*;  
 numeLabel.setBounds*(*10, 123, 74, 36*)*;  
 this.getContentPane*()*.add*(*numeLabel*)*;  
  
 adresaLabel = new JLabel*(*"ADRESA:"*)*;  
 adresaLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 adresaLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 18*))*;  
 adresaLabel.setBounds*(*10, 172, 86, 36*)*;  
 this.getContentPane*()*.add*(*adresaLabel*)*;  
  
 dateLabel = new JLabel*(*"DATE CLIENT"*)*;  
 dateLabel.setHorizontalAlignment*(*SwingConstants.*CENTER)*;  
 dateLabel.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 18*))*;  
 dateLabel.setBounds*(*109, 21, 156, 46*)*;  
 this.getContentPane*()*.add*(*dateLabel*)*;  
  
 // Tabel  
 clientTable = new JTable*()*;  
 this.tableScroll = new JScrollPane*(*this.clientTable, JScrollPane.*VERTICAL\_SCROLLBAR\_ALWAYS*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_ALWAYS)*;  
 this.tableScroll.setBounds*(*305, 77, 687, 389*)*;  
 this.tableScroll.setViewportView*(*this.clientTable*)*;  
 this.getContentPane*()*.add*(*this.tableScroll*)*;  
  
 // Casete text  
 idTextField = new JTextField*()*;  
 idTextField.setBounds*(*109, 77, 156, 36*)*;  
 this.getContentPane*()*.add*(*idTextField*)*;  
 idTextField.setColumns*(*10*)*;  
  
 numeTextField = new JTextField*()*;  
 numeTextField.setColumns*(*10*)*;  
 numeTextField.setBounds*(*109, 126, 156, 36*)*;  
 this.getContentPane*()*.add*(*numeTextField*)*;  
  
 adresaTextField = new JTextField*()*;  
 adresaTextField.setColumns*(*10*)*;  
 adresaTextField.setBounds*(*109, 172, 156, 36*)*;  
 this.getContentPane*()*.add*(*adresaTextField*)*;  
  
 // Butoane  
 adaugaButton = new JButton*(*"ADAUGA"*)*;  
 adaugaButton.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 18*))*;  
 adaugaButton.setBounds*(*109, 239, 156, 38*)*;  
 this.getContentPane*()*.add*(*adaugaButton*)*;  
  
 stergeButton = new JButton*(*"STERGE"*)*;  
 stergeButton.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 18*))*;  
 stergeButton.setBounds*(*109, 287, 156, 38*)*;  
 this.getContentPane*()*.add*(*stergeButton*)*;  
  
 modificaButton = new JButton*(*"MODIFICA"*)*;  
 modificaButton.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 18*))*;  
 modificaButton.setBounds*(*109, 335, 156, 38*)*;  
 this.getContentPane*()*.add*(*modificaButton*)*;  
  
 afisareButton = new JButton*(*"AFISARE"*)*;  
 afisareButton.setFont*(*new Font*(*"Times New Roman", Font.*PLAIN*, 18*))*;  
 afisareButton.setBounds*(*109, 383, 156, 38*)*;  
 this.getContentPane*()*.add*(*afisareButton*)*;  
  
 // Afisare fereastra actuala  
 this.setVisible*(*true*)*;  
*}*

*/\*\*  
 \* Metoda de adaugare a unui listener pe butonul de adaugare  
 \*  
 \** ***@param*** *actionListener actionListener  
 \*/*public void addAdaugareListener*(*ActionListener actionListener*) {* this.adaugaButton.addActionListener*(*actionListener*)*;  
*}*

**16.** **Clasa ProductView** – impementează elementele interfeței grafice pentru Product

– conține atributele necesare în casete text, a etichetelor corepunzătoare acestora, precum și a butoanelor de adăugare, modificare, ștergere și afișare.

În constructor vom poziționa toate elementele interfeței, pe lângă metodele de get() și set() pentru castele text, avem metode de adăugare a unor listeners pentru a verifica dacă unul dintre butoane este apăsat, caz în care se trimite un apel către controller pentru a afișa rezulatul corepunzător.

**17.** **Clasa OrderView** – impementează elementele interfeței grafice pentru Order

– conține atributele necesare în casete text, a etichetelor corepunzătoare acestora, precum și a butoanelor de adăugare și afișare.

În constructor vom poziționa toate elementele interfeței, pe lângă metodele de get() și set() pentru castele text, avem metode de adăugare a unor listeners pentru a verifica dacă unul dintre butoane este apăsat, caz în care se trimite un apel către controller pentru a afișa rezulatul corepunzător.

**18.** **Clasa ClientController** – primește un apel când este apăsat un buton de pe interfață, iar după efectuarea operației corespunzătoare butonului, returnează rezultatul în tabel (butonul AFISARE) sau afișează un mesaj corespunzător (restul butoanelor).

Constructorul primește ca parametru view-ul și clientBLL, iar acesta are rolul de a lega butoanele de metoda care apelează operația corepunzătoare.

*/\*\*  
 \* Constructor pentru clasa ClientController  
 \*  
 \** ***@param*** *view view-ul pentru clienti  
 \** ***@param*** *clientBLL bll-ul pentru clienti  
 \*/*public ClientController*(*ClientView view, ClientBLL clientBLL*) {* this.view = view;  
 this.clientBLL = clientBLL;  
  
 this.view.addAfisareListener*(*new AfisareListener*())*;  
 this.view.addAdaugareListener*(*new AdaugareListener*())*;  
 this.view.addStergereListener*(*new StergereListener*())*;  
 this.view.addModificareListener*(*new ModificareListener*())*;  
*}*

Apoi avem metodele care implementeaza interfața ActionListener prin care se verifică inputul primit după apăsarea unui buton.

*/\*\*  
 \* Clasa pentru implementarea inserarii datelor in tabela Client in interfata corespunzatoare  
 \*/*class AdaugareListener implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*) {* try *{* // Preluare date de pe interfata  
 int id = Integer.*parseInt(*view.getIdTextField*()*.getText*())*;  
 String nume = view.getNumeTextField*()*.getText*()*;  
 String adresa = view.getAdresaTextField*()*.getText*()*;  
  
 Client client = new Client*(*id, nume, adresa*)*; // creare client  
  
 clientBLL.insert*(*client*)*; // inserare client  
  
 JOptionPane.*showMessageDialog(*null, "Client adaugat cu succes!"*)*;  
 *}* catch *(*NumberFormatException ex*) {* JOptionPane.*showMessageDialog(*null, "ID invalid!"*)*;  
 *}* catch *(*Exception ex*) {* JOptionPane.*showMessageDialog(*null, ex.getMessage*())*;  
 *}  
 }  
}*

Pentru a afișa tabelul se folosește tehnica de reflexie astfel:

*/\*\*  
 \* Metoda pentru creare capurilor de tabel  
 \*  
 \** ***@param*** *objectList lista de obiecte din care se va crea capul de tabel  
 \** ***@param*** *table tabelul obtinut  
 \*/*private void reflectionTableHeaders*(*List*<*Object*>* objectList, JTable table*) {* DefaultTableModel tableModel = new DefaultTableModel*()*;  
  
 Object obj = objectList.get*(*0*)*;  
  
 for *(*Field field : obj.getClass*()*.getDeclaredFields*()) {* field.setAccessible*(*true*)*;  
  
 try *{* // Adaugam drept coloana in tabel fiecare atribut al obiectului  
 tableModel.addColumn*(*field.getName*())*;  
 *}* catch *(*IllegalArgumentException ex*) {* ex.printStackTrace*()*;  
 *}  
 }* for *(*Object o : objectList*) {* // Vom avea cate o lista de valori pentru atribute pentru fiecare obiect din lista primita ca parametru  
 List*<*Object*>* valoriAtribute = new ArrayList*<>()*;  
  
 for *(*Field field : o.getClass*()*.getDeclaredFields*()) {* field.setAccessible*(*true*)*;  
 Object value;  
 try *{* // Luam valorile pentru fiecare atribut  
 value = field.get*(*o*)*;  
  
 // Adaugam valorile atributelor intr-o lista  
 valoriAtribute.add*(*value*)*;  
 *}* catch *(*IllegalArgumentException | IllegalAccessException ex*) {* ex.printStackTrace*()*;  
 *}  
 }* // Convertim lista de obiecte in vector de obiecte  
 tableModel.addRow*(*valoriAtribute.toArray*())*;  
 *}* table.setModel*(*tableModel*)*; // actualizam tabelul  
*}  
  
/\*\*  
 \* Clasa pentru implementarea afisarii datelor din tabela Client in interfata corespunzatoare  
 \*/*class AfisareListener implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*) {* try *{* List*<*Client*>* clients = clientBLL.findAll*()*; // lista clientilor din tabela Client  
  
 List*<*Object*>* objects = new ArrayList*<>()*; // lista de obiecte  
  
 for *(*Client c : clients*)* // cast manual pentru lista de clienti  
 objects.add*(*c*)*;  
  
 reflectionTableHeaders*(*objects, view.getTable*())*; // adaugam in tabel  
  
 *}* catch *(*NumberFormatException exception*) {* exception.printStackTrace*()*;  
 *}  
 }  
}*

**19.** **Clasa ProductController** – primește un apel când este apăsat un buton de pe interfață, iar după efectuarea operației corespunzătoare butonului, returnează rezultatul în tabel (butonul AFISARE) sau afișează un mesaj corespunzător (restul butoanelor).

Constructorul primește ca parametru view-ul și productBLL, iar acesta are rolul de a lega butoanele de metoda care apelează operația corepunzătoare.

**20.** **Clasa OrderController** – primește un apel când este apăsat un buton de pe interfață, iar după efectuarea operației corespunzătoare butonului, returnează rezultatul în tabel (butonul AFISARE) sau afișează un mesaj corespunzător (buton ADAUGARE).

Constructorul primește ca parametru view-ul și orderBLL, iar acesta are rolul de a lega butoanele de metoda care apelează operația corepunzătoare.

Pentruu adăugarea unei comenzi se selectează un client și un produs din tabelele respective și se introduce ID-ul, precum și cantitatea.

*/\*\*  
 \* Clasa pentru implementarea inserarii datelor in tabela Order in interfata corespunzatoare  
 \*/*class AdaugareListener implements ActionListener *{* @Override  
 public void actionPerformed*(*ActionEvent e*) {* try *{* // Preluare date de pe interfata  
 int id = Integer.*parseInt(*view.getIdTextField*()*.getText*())*;  
 int cantitate = Integer.*parseInt(*view.getCantitateTextField*()*.getText*())*;  
  
 // Preluare randuri selectate din tabelele Client si Product  
 int randSelectatClienti = view.getClientTable*()*.getSelectedRow*()*;  
 int randSelectatProduse = view.getProdusTable*()*.getSelectedRow*()*;  
  
 String idClientString = view.getClientTable*()*.getModel*()*.getValueAt*(*randSelectatClienti, 0*)*.toString*()*;  
 String idProdusString = view.getProdusTable*()*.getModel*()*.getValueAt*(*randSelectatProduse, 0*)*.toString*()*;  
  
 // Conversie date  
 int idClient = Integer.*parseInt(*idClientString*)*;  
 int idProdus = Integer.*parseInt(*idProdusString*)*;  
  
 Order order = new Order*(*id, idClient, idProdus, cantitate*)*; // creare comanda  
  
 orderBLL.insert*(*order*)*; // inserare comanda  
  
 // Afisare in fisier  
 String toFile = "ID: " + id + "\nID Client: " + idClient + "\nID Produs: " + idProdus + "\nCantitate: " + cantitate;  
 OrderFile.*writeToFile(*toFile, "order" + order.getId*()* + ".txt"*)*;  
  
 JOptionPane.*showMessageDialog(*null, "Comanda adaugata cu succes!"*)*;  
 *}* catch *(*NumberFormatException ex*) {* JOptionPane.*showMessageDialog(*null, "ID / Cantitate invalida!"*)*;  
 *}* catch *(*Exception ex*) {* JOptionPane.*showMessageDialog(*null, ex.getMessage*())*;  
 *}  
 }  
}*

**21.** **Clasa ConnectionFactory** – realizează legătura cu baza de date corepunzătoare.

private static final String *DBURL* = "jdbc:mysql://localhost:3306/depozit";  
private static final String *USER* = "";  
private static final String *PASS* = "";  
  
private static ConnectionFactory *singleInstance* = new ConnectionFactory*()*;  
  
*/\*\*  
 \* Constructor pentru clasa ConnectionFactory  
 \*  
 \** ***@return*** *conexiunea creata  
 \*/*private Connection createConnection*() {* Connection connection = null;  
 try *{* connection = DriverManager.*getConnection(DBURL*, *USER*, *PASS)*;  
 *}* catch *(*SQLException e*) {* e.printStackTrace*()*;  
 *}* return connection;  
*}  
  
/\*\*  
 \* Metoda pentru obtinerea conexiunii active  
 \*  
 \** ***@return*** *conexiunea ceruta  
 \*/*public static Connection getConnection*() {* return *singleInstance*.createConnection*()*;  
*}  
  
/\*\*  
 \* Metoda pentru inchiderea conexiunii  
 \*  
 \** ***@param*** *connection conexiunea cu baza de date  
 \*/*public static void close*(*Connection connection*) {* if *(*connection != null*) {* try *{* connection.close*()*;  
 *}* catch *(*SQLException e*) {* e.printStackTrace*()*;  
 *}  
 }*

**22. Clasa OrderFile** – conține o metodă care scrie în fișier datele fiecărei comenzi.

*/\*\*  
 \* Metoda in care se va scrie in fisier fiecare comanda adaugata  
 \** ***@param*** *content ce trebuie scris in fisier  
 \** ***@param*** *file fisierul in care trebuie scris   
 \*/*public static void writeToFile*(*String content, String file*) {* try *{* String filePath = "D:\\Facultate\\UTCN\\An II\\Semestrul 2\\Tehnici de programare\\Laboratoare\\PT2022\_30228\_Gavrilescu\_AndreeaLavinia\_3\\src\\main\\resources\\files\\" + file;  
 BufferedWriter bw = new BufferedWriter*(*new FileWriter*(*filePath*))*;  
 bw.write*(*content*)*;  
 bw.close*()*;  
 *}* catch *(*IOException ex*) {* ex.printStackTrace*()*;  
 *}  
}*

**23. Clasa Main** – conține atributele necesare controllere-lor astfel încât să realizeze legătura între interfață și logică.

Validator*<*Client*>* clientValidator = new ClientValidator*()*;  
ClientDAO clientDAO = new ClientDAO*()*;  
ClientBLL clientBLL = new ClientBLL*(*clientValidator, clientDAO*)*;  
  
ClientView clientView = new ClientView*()*;  
ClientController clientController = new ClientController*(*clientView, clientBLL*)*;  
  
Validator*<*Product*>* productValidator = new ProductValidator*()*;  
ProductDAO productDAO = new ProductDAO*()*;  
ProductBLL productBLL = new ProductBLL*(*productValidator, productDAO*)*;  
  
ProductView productView = new ProductView*()*;  
ProductController productController = new ProductController*(*productView, productBLL*)*;  
  
Validator*<*Order*>* orderValidator = new OrderValidator*()*;  
OrderDAO orderDAO = new OrderDAO*()*;  
OrderBLL orderBLL = new OrderBLL*(*orderValidator, clientDAO, productDAO, orderDAO*)*;  
  
OrderView orderView = new OrderView*()*;  
OrderController orderController = new OrderController*(*orderView, clientBLL, productBLL, orderBLL*)*;

Rezultate

Pentru fiecare comandă adăugată se va genera un fișier care conține infomațiile corepunzătoare acesteia.

Exemplu:

Order2.txt

ID: 2  
ID Client: 2  
ID Produs: 4  
Cantitate: 1

Concluzii

Acest proiect mi-a consolidat cunoștințele teoretice și practice de Java pe care le-am învățat în semestrul 1 prin volumul de lucru mult mai intens și avansat pe care l-am depus în scopul finalizării acestei teme.

Ca dezvoltare ulterioară, se mai poate îmbunătăți aspectul interfeței ca să arate mai interesant, se mai poate adăuga tot felul de butoane de back pentru a naviga mai ușor între interfețe, precum și butoane care să asigure o conexiune mai buna între acestea, putând fi utlizată eventual pentru comenzi online.

Bibliografie

* <https://dsrl.eu/courses/pt/materials/A3_Support_Presentation.pdf>
* <https://gitlab.com/utcn_dsrl/pt-layered-architecture>
* <https://gitlab.com/utcn_dsrl/pt-reflection-example>