

# DOCUMENTATIE

## TEMA 3

NUME STUDENT: Ureche Simona Elena  
GRUPA: 30224

# CUPRINS

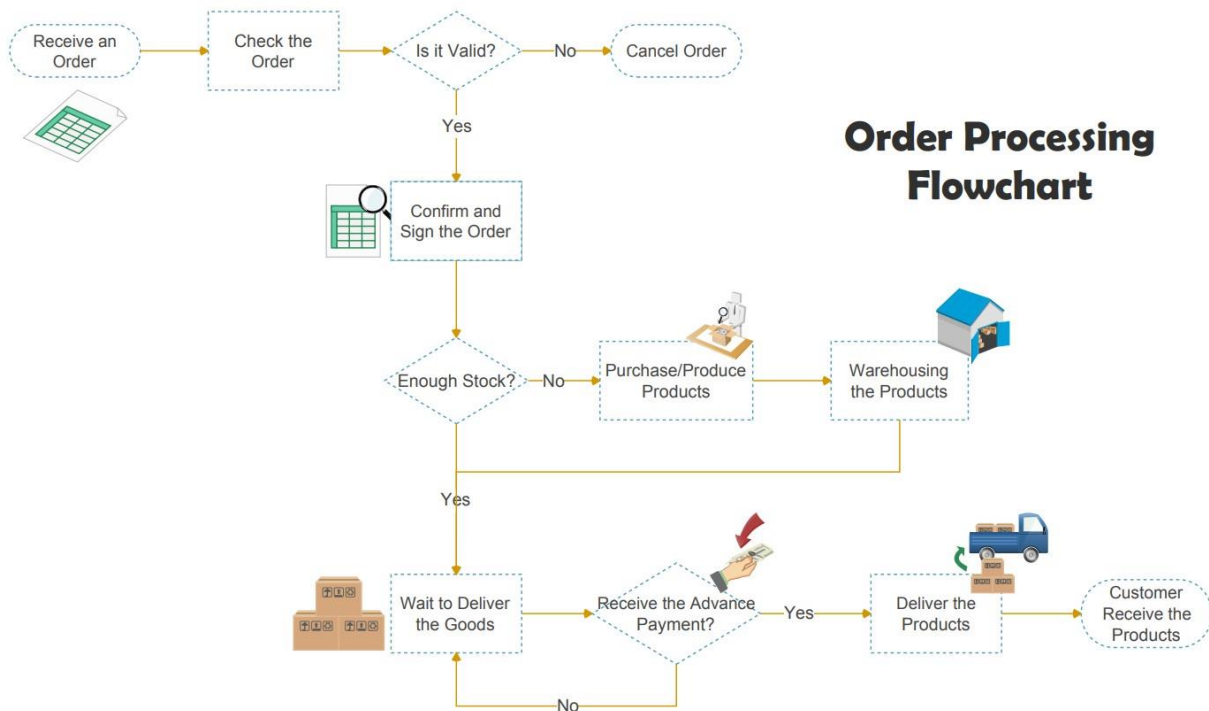
1.	Obiectivul temei.....	3
2.	Analiza problemei, modelare, scenarii, cazuri de utilizare .....	4
3.	Proiectare .....	5
4.	Implementare .....	13
5.	Concluzii.....	18
6.	Bibliografie .....	19

## 1. Obiectivul temei

- (i) *Obiectivul principal*  
(ii) *Dezvoltarea unei aplicații cu interfață grafică dedicată managementului coșilor pentru procesarea comenzilor clientilor pentru un depozit. Sunt folosite baze de date relationale pentru a stoca produsele, clientii si comenzile.*

<i>Obiectiv secundar</i>	<i>Descriere</i>	<i>Capitol</i>
<i>Cerințe funcționale/nonfuncționale</i>	Definesc comportamentul și capacitățile sistemului. Acestea pot include operațiuni specifice, funcționalități, servicii sau caracteristici pe care sistemul trebuie să le ofere pentru a satisface nevoile utilizatorilor și a atinge obiectivele stabilite. Scenariile oferă o perspectivă detaliată asupra modului în care utilizatorii și sistemul colaborează pentru a realiza sarcini.	2
<i>Proiectarea orientată pe obiecte(POO)</i>	Prezentarea arhitecturii generale a aplicației, evidențiind modul în care conceptele OOP au fost aplicate pentru a organiza și structura codul.	3
<i>Împărțirea în pachete și clase</i>	Modul în care clasele și alte resurse sunt grupate în pachete logice pentru a organiza și gestiona proiectul.	3
<i>Concepte utilizate</i>	Utilizate în cadrul aplicației pentru a rezolva diferite probleme sau sarcini.	3
<i>Interfete grafice definite</i>	Folosirea tehnicii Model View Controller pentru realizarea unui Graphical User Interface.	3
<i>Descrierea claselor și metodelor</i>	Semnăturile și descrierea claselor și metodelor relevante din cadrul aplicației de management al coșilor.	4
<i>Implementarea interfeței utilizator</i>	Modul în care interfața utilizator a fost proiectată și implementată pentru a facilita interacțiunea utilizatorului cu manipularea clientilor, produselor si a comenzilor.	4

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



### Scenariu de Utilizare:

Actor Principal: Utilizatorul

#### Cerințe Funcționale:

- Aplicația de simulare ar trebui să permit unui angajat sa aduce un nou client.
- Aplicația de simulare ar trebui să permit unui angajat sa aduce un nou produs.
- Aplicația de simulare ar trebui să permit unui angajat sa aduce o noua comanda.

#### Cerințe Non-Funcționale:

- Aplicația de simulare trebuie să fie intuitivă și ușor de utilizat de către utilizator.
- Aplicația de simulare trebuie să poată gestiona eficient un număr mare de comenzi.
- Aplicația de simulare trebuie să fie robustă și să nu fie predispusă la căderi sau comportamente neașteptate.

#### Scenariul Principal de Succes:

- Utilizatorul selectează opțiunea pentru a adauga un nou produs:
- Aplicația va afișa un formular în care sunt detaliile produsului care trebuie introdus.
- Angajatul introduce numele produsului, pretul și stocul curent.
- Dacă datele sunt valide, aplicația afișează un mesaj de confirmare și stochează datele produsului în baza de date.

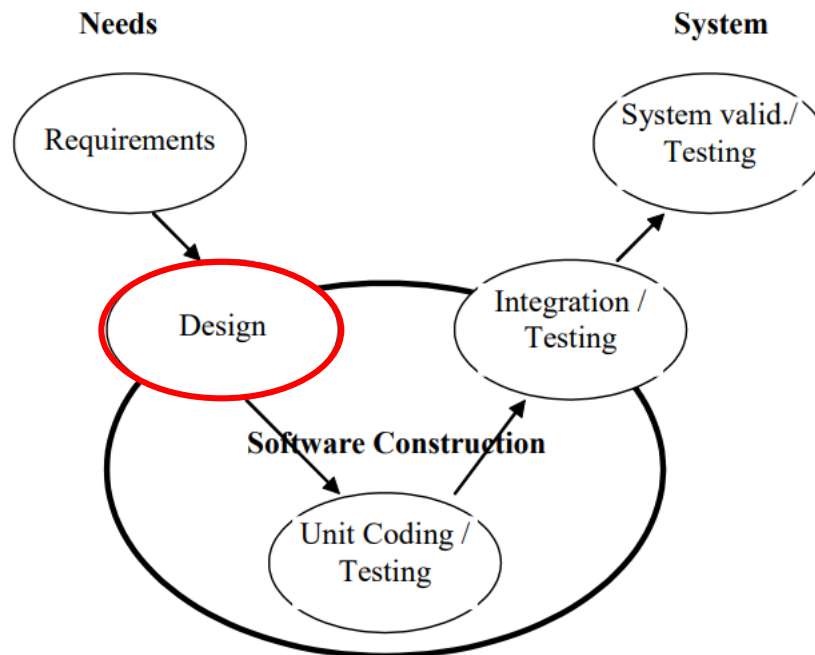
Secvență Alternativă: Date Invalide

### 3. Proiectare

- Utilizatorul introduce valori invalide pentru stocul produsului.
- Aplicatia afisează mesaj de eroare și îi permite utilizatorului să introducă alte date valide.
- Scenariul revine la pasul 3.

(i) **Decizii de proiectare**

Aplicația dedicată manipulării unui sistem eficient de management al cozilor pentru procesarea comenzilor clientilor pentru un depozit folosind conceptele de programare orientată pe obiecte (OOP) pentru a organiza și structura codul într-un mod coerent și eficient. Vom separa atât componentele logice cât și modelul de date de interfață de control, ceea ce face mai ușoară gestionarea și întreținerea codului.



(ii) **Împărțirea în pachete**

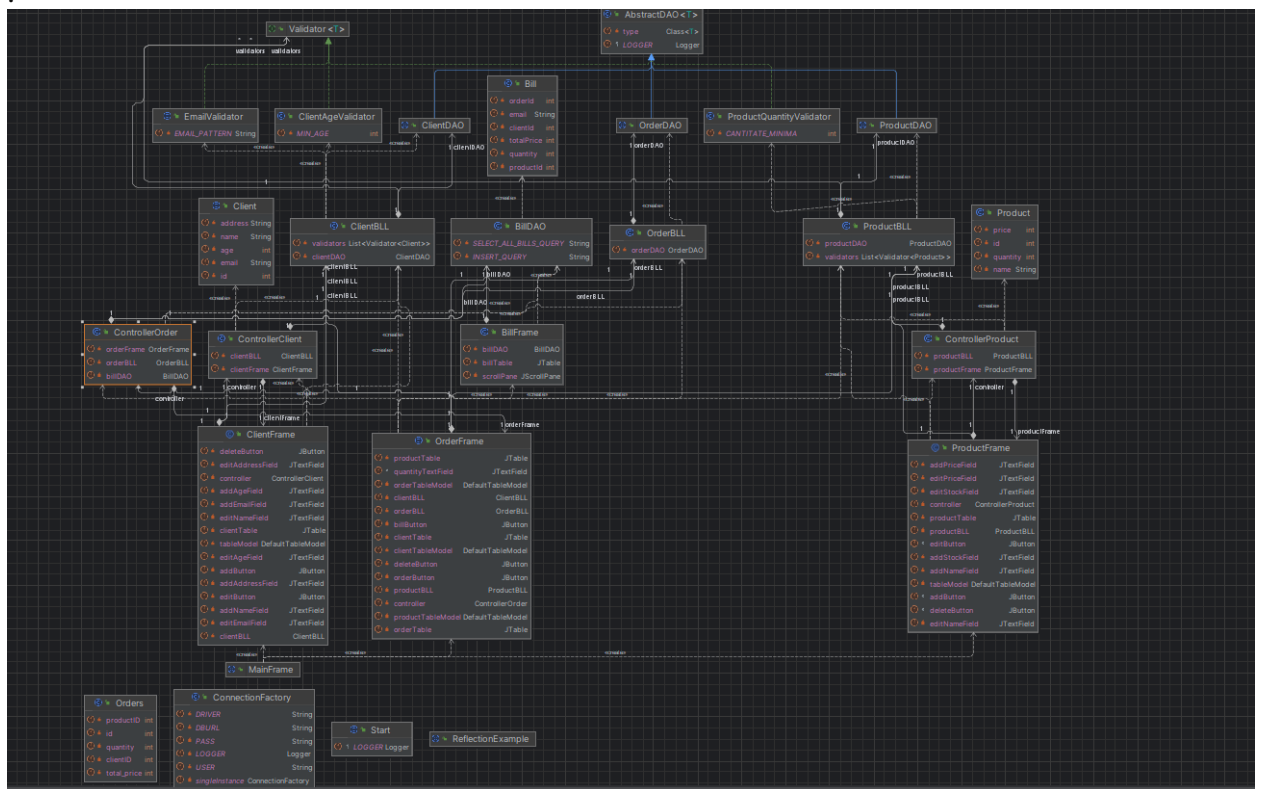
- Pachetul bll (Business Logic Layer): Acest pachet conține o parte din logica aplicației. În interiorul său, se pot găsi clase care definesc operațiile și regulile de afaceri care guvernează modul în care datele sunt manipulate în aplicație, inclusiv validarea datelor folosind un validator.

-Pachetul connection: Pachetul connection este responsabil de gestionarea conexiunii la baza de date. Aici se găsesc clase care facilitează crearea și închiderea conexiunilor cu baza de date.

- Pachetul dao (Data Access Object): Acest pachet conține clase care oferă o interfață pentru accesul la date în baza de date. Aceste clase sunt responsabile pentru efectuarea operațiilor CRUD (Create, Read, Update, Delete) și maparea datelor din baza de date în obiecte Java și invers.

- Pachetul model: Pachetul model conține clase care reprezintă entitățile sau obiectele de date din aplicație. Aceste clase reflectă structura datelor din baza de date și sunt utilizate pentru a stoca și manipula date în aplicație.

- Pachetul presentation: Acest pachet conține clasele care definesc interfața cu utilizatorul (UI) a aplicației. Aici sunt definite ferestrele, butoanele și alte elemente de interfață grafică necesare pentru interacțiunea utilizatorului cu aplicația.



### (iii) Împărțirea în clase

- ClientAgeValidator : Această clasă validează vârsta unui client în funcție de anumite reguli prestabilite, oferind metode pentru verificarea dacă vârsta este într-un anumit interval sau respectă alte condiții.

- EmailValidator: Clasa EmailValidator se ocupă de validarea adreselor de email, verificând dacă acestea sunt într-un format corect și respectă anumite reguli de validare a adresei de email.
- ProductQuantityValidator: Această clasă este responsabilă pentru validarea cantității unui produs, asigurându-se că cantitatea introdusă este validă din punct de vedere al stocului disponibil.
- ClientBLL: ClientBLL conține logica asociată entității "Client". Aici pot fi definite operații precum adăugarea unui client, actualizarea datelor clientului sau obținerea informațiilor despre un anumit client.
- OrderBLL: Clasa OrderBLL conține logica asociată entității "Order". Aceasta poate include operații precum plasarea unei comenzi, calcularea totalului comenzii sau gestionarea stării unei comenzi.
- ProductBLL: Această clasă conține logica asociată entității "Product". Poate include operații precum adăugarea unui produs, actualizarea datelor produsului sau obținerea informațiilor despre un produs.
- ConnectionFactory: Clasa ConnectionFactory facilitează gestionarea conexiunii la baza de date. Oferă metode pentru obținerea unei conexiuni și pentru închiderea acesteia după utilizare.
- AbstractDAO: AbstractDAO este o clasă abstractă care oferă operații generice pentru accesul la date în baza de date. Implementările concrete ale acestui DAO pot să furnizeze metode pentru a efectua operații CRUD pe datele specifice unei entități.
- BillDAO: BillDAO este o clasă care implementează operațiile specifice pentru accesul la date pentru entitatea "Bill". Aceasta poate include metode pentru inserarea, actualizarea, ștergerea și interogarea datelor facturilor din baza de date.
- ClientDAO: ClientDAO este o clasă care implementează operațiile specifice pentru accesul la date pentru entitatea "Client". Aceasta poate include metode pentru gestionarea datelor despre clienți în baza de date.
- OrderDAO: OrderDAO este o clasă care implementează operațiile specifice pentru accesul la date pentru entitatea "Order". Aceasta poate include metode pentru gestionarea datelor despre comenzile efectuate în baza de date.
- ProductDAO: ProductDAO este o clasă care implementează operațiile specifice pentru accesul la date pentru entitatea "Product". Aceasta poate include metode pentru gestionarea datelor despre produse în baza de date.
- Bill: Clasa Bill reprezintă entitatea "Bill" din aplicație, care conține informații despre o factură, cum ar fi data emiterii, suma totală etc.

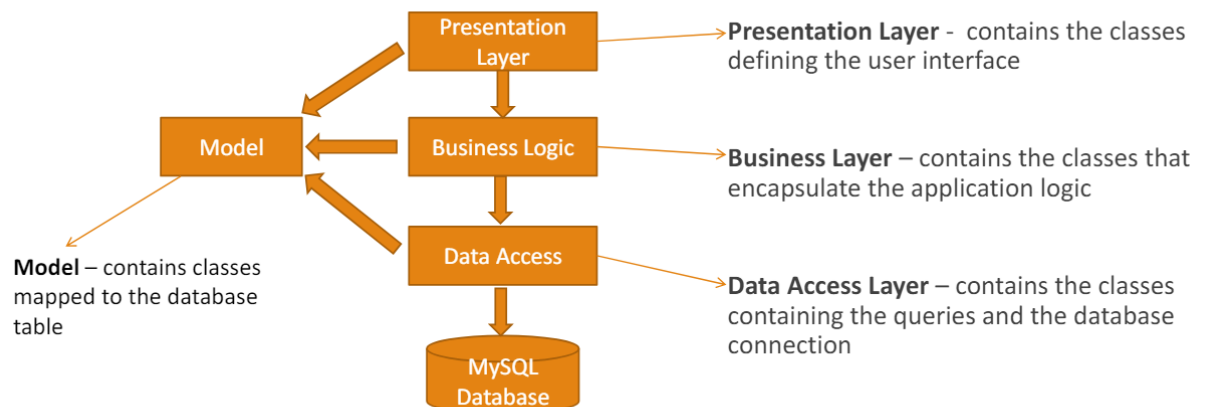
- Client: Clasa Client reprezintă entitatea "Client" din aplicație, care poate conține informații despre un client, cum ar fi nume, adresă, email etc.
- Order: Clasa Order reprezintă entitatea "Order" din aplicație, care conține informații despre o comandă, cum ar fi produsele comandate, cantitățile, data etc.
- Product: Clasa Product reprezintă entitatea "Product" din aplicație, care conține informații despre un produs, cum ar fi nume, preț, stoc etc.
- BillFrame: Această clasă definește interfața grafică pentru gestionarea facturilor în aplicație.
- ClientFrame: Clasa ClientFrame definește interfața grafică pentru gestionarea clienților în aplicație.
- ControllerClient: ControllerClient este o clasă care gestionează interacțiunea dintre interfața grafică pentru clienți și logica asociată acestora.
- ControllerOrder: ControllerOrder este o clasă care gestionează interacțiunea dintre interfața grafică pentru comenzi și logica asociată acestora.
- ControllerProduct: ControllerProduct este o clasă care gestionează interacțiunea dintre interfața grafică pentru produse și logica asociată acestora.
- MainFrame: MainFrame este clasa principală care inițializează și afișează fereastra principală a aplicației.
- OrderFrame: OrderFrame este clasa care definește interfața grafică pentru gestionarea comenzilor în aplicație.
- ProductFrame: ProductFrame este clasa care definește interfața grafică pentru gestionarea produselor în aplicație..

#### **(iv) Concepte utilizate**

- Vom folosi ArrayList, la un moment dat, pentru a stoca clientii, produsele si comenzile, iar apoi toate aceste date se vor transmite mai departe in entitatile care se ocupa cu afisarea rezultatelor finale. O structura de date destul de usor de manevrat care contine mai multe metode ce ne vor ajuta in realizarea obiectivului propus.
- Utilizarea Generics în Reflection Techniques: Am definit clasa `AbstractDAO<T>` ca o clasă generică, care poate fi specializată pentru diferite tipuri de entități (cum ar fi `Client`, `Order`, `Product`, `Bill`, etc.). Aceasta îți permite să folosești același cod pentru a accesa și manipula datele pentru diverse tipuri de obiecte, fără a fi nevoie să rescrii logica de acces la date pentru fiecare entitate în parte.



- De asemenea am folosit metode generice în cadrul clasei `AbstractDAO` pentru a implementa operații comune, cum ar fi inserarea, actualizarea, ștergerea și găsirea obiectelor din baza de date. Acest lucru îți oferă flexibilitate în tratarea diferitelor tipuri de obiecte, fără a fi nevoie să scrii metode separate pentru fiecare entitate.

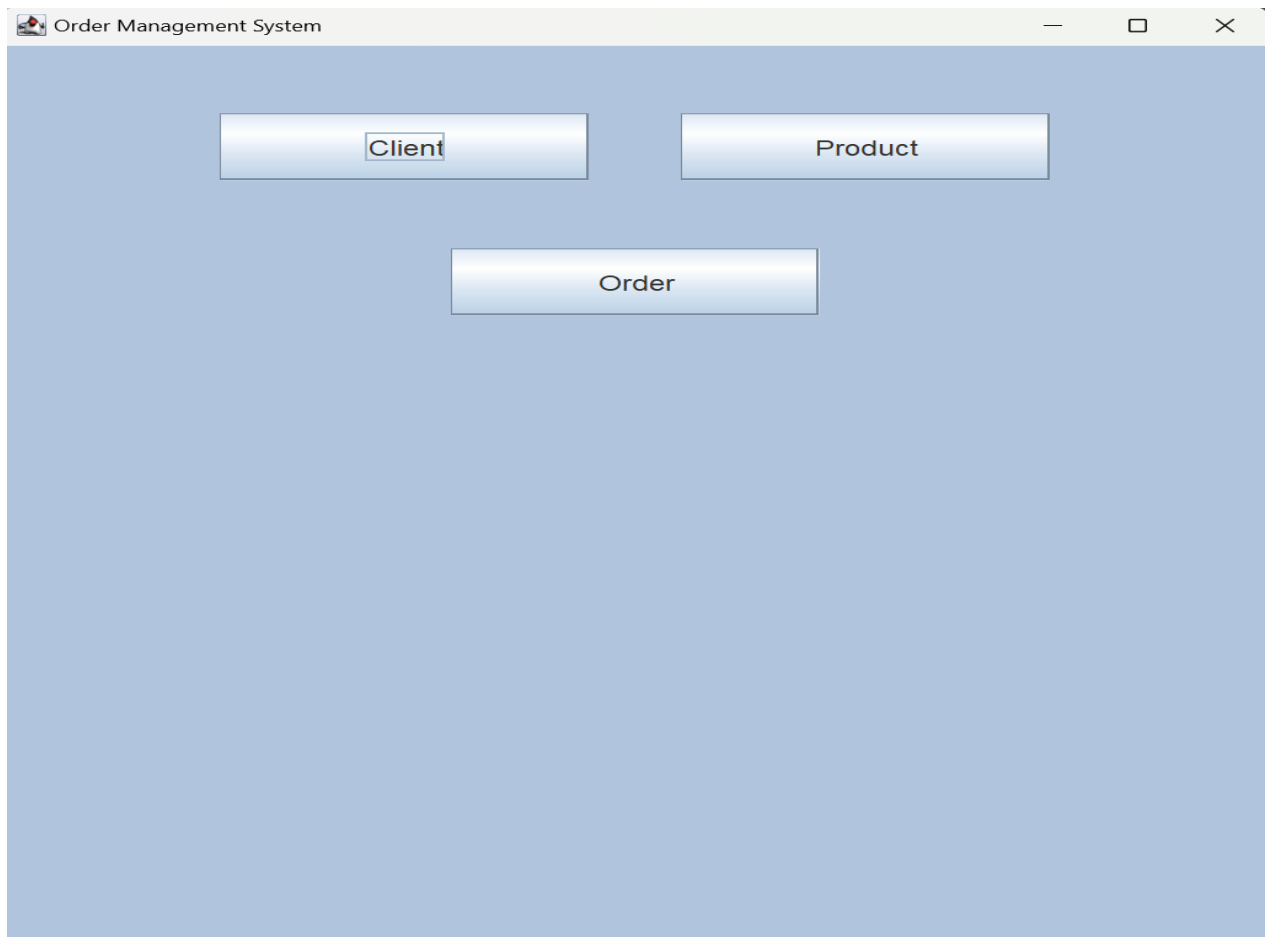


- Utilizarea Reflection Techniques: Am utilizat Reflection Techniques pentru a genera interogările SQL dinamice în funcție de structura și numele câmpurilor din clasele de entități. Astfel, interogările SQL sunt adaptabile și care să funcționeze cu orice tip de entitate, fără a fi nevoie să specifici manual numele câmpurilor în interogările tale.

- De asemenea am utilizat Reflection Techniques pentru a instanția dinamic obiectele la runtime în cadrul metodelor `createObjects(ResultSet resultSet)` și `setGeneratedId(T t, int generatedId)` din clasa `AbstractDAO`.

#### (v) Interfața grafică

- Contine fereastra principală a aplicației, care are butoane pentru navigarea către gestionarea clienților, comenzilor și produselor.



- Fereastra pentru gestionarea clienților afișează o listă a clienților existenți și butoane pentru adăugarea, editarea și ștergerea clienților. Permite introducerea informațiilor despre un nou client sau editarea informațiilor unui client existent.

Client Window

id	name	address	email	age
1	Ureche	Parang 15	ureche.simona@yahoo...	20
2	Repede	Floresti 2	repede.damaris@yahoo...	21
3	Ureche	Garli 1082	ureche.elena@yahoo...	40
9	Bumbu	Valea Caselor 108	bumbu.gabriela@yahoo...	22
10	Macavei	Granicerilor 6	macavei.ai@yahoo.com	25

Add Client

Name:   
Address:   
Email:   
Age:

Edit Client

Name:   
Address:   
Email:   
Age:

Delete Client

Add Client Edit Client Delete Client

- Fereastra pentru gestionarea comenzilor afiseaza o listă a comenzilor existente și butoane pentru adăugarea, editarea și ștergerea comenzilor. Permite asocierea comenzilor cu clienții existenți și produsele disponibile.

Commands window

Quantity:

id	name	address	email	age
1	Ureche	Parang 15	ureche.sim...	20
2	Repede	Floresti 2	repede.da...	21
3	Ureche	Garli 1082	ureche.ele...	40
9	Bumbu	Valea Case...	bumbu.gab...	22
10	Macavel	Granicerilor 6	macavel.al...	25

id	quantity	price	name
1	19	1222	aspirator
2	10	30	ciocolata
4	7	1500	masina de spalat
5	24	220	uscator
6	99	199	ochelari

id	clientID	productID	quantity	total_price
12	9	2	9	30
13	10	1	19	1222

Order Delete BILL

- Fereastra pentru gestionarea produselor (ProductFrame) afișează o listă a produselor existente și butoane pentru adăugarea, editarea și ștergerea produselor. Permite introducerea informațiilor despre un nou produs sau editarea informațiilor unui produs existent.

Commands window

id	quantity	price	name
1	19	1222	aspirator
2	10	30	ciocolata
4	7	1500	masina de spalat
5	24	220	uscator
6	99	199	ochelari

**Add**

Name:

Price:

Quantity:

**Edit**

Name:

Price:

Quantity:

**Delete**

Add Product Edit Product Delete Product

## (vi) Baza de date

Baza de serveste drept fundamentul pentru stocarea și gestionarea datelor referitoare la clienți, comenzi și produse. Ea este structurată în jurul a patru tabele principale, fiecare reprezentând un aspect specific al informațiilor gestionate:

### 1. Tabela Client:

- Această tabelă stochează informații despre clienți, cum ar fi numele, adresa, adresa de email și alte detalii relevante.
- Fiecare înregistrare în această tabelă reprezintă un client distinct, iar fiecare client poate avea asociate una sau mai multe comenzi.

### 2. Tabela Order :

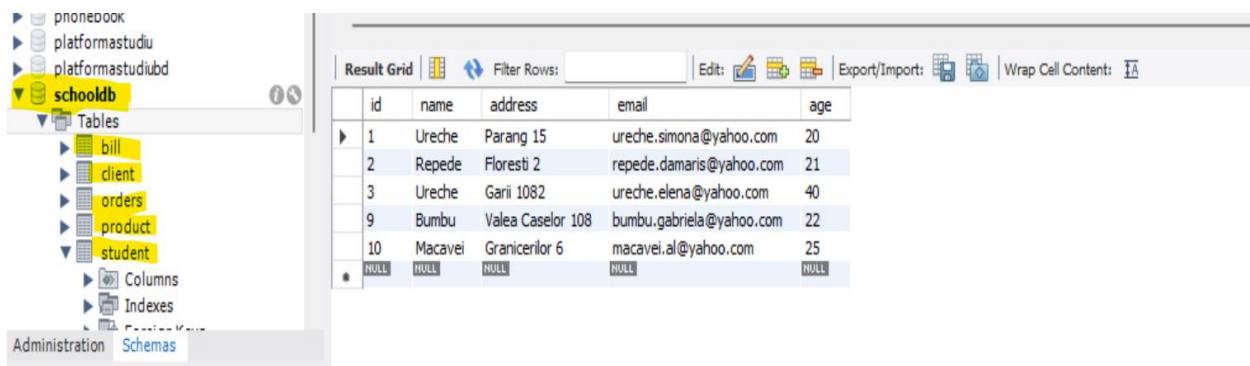
- Tabela Order stochează detalii despre comenzile plasate de clienți, inclusiv data comenzii și orice alte informații specifice comenzii.
- Fiecare înregistrare în această tabelă reprezintă o comandă distinctă, iar fiecare comandă este asociată cu un client și conține informații despre produsele achiziționate.

### 3. Tabela Product :

- Această tabelă conține informații despre produsele disponibile pentru achiziție, cum ar fi numele, prețul și stocul disponibil.
- Fiecare înregistrare în această tabelă reprezintă un produs distinct, iar aceste produse pot fi asociate cu una sau mai multe comenzi.

### 4. Tabela Bill (Facturi):

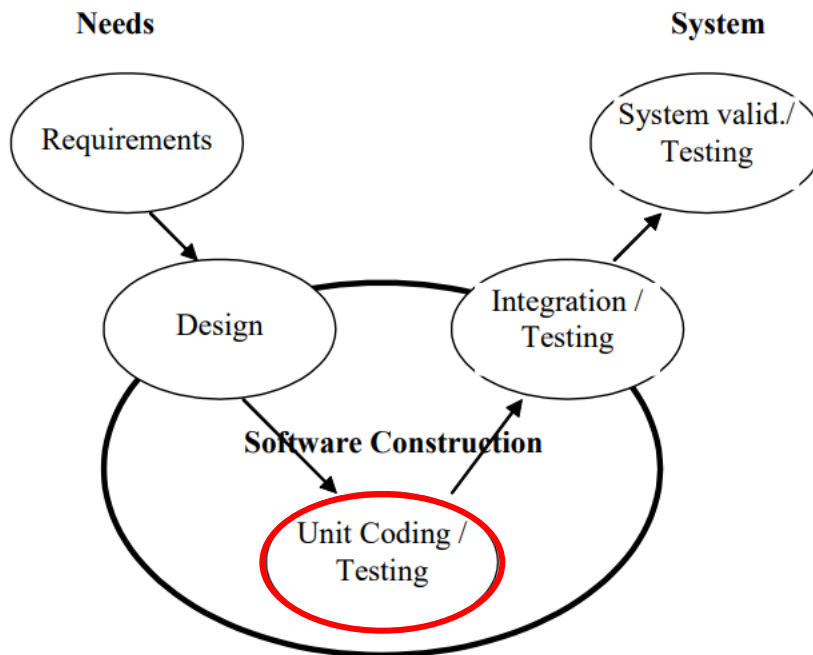
- Tabela Bill stochează informații despre facturile generate pentru fiecare comandă, inclusiv totalul facturat și alte detalii legate de plată.
- Fiecare înregistrare în această tabelă reprezintă o factură distinctă asociată cu o comandă specifică și poate fi folosită pentru urmărirea și gestionarea tranzacțiilor financiare.



The screenshot displays a database management interface. On the left, a tree view shows the database structure with 'schooldb' selected. The main area shows a 'Result Grid' for the 'client' table. The table has five columns: 'id', 'name', 'address', 'email', and 'age'. It contains 10 rows of data and a final row with NULL values.

id	name	address	email	age
1	Ureche	Parang 15	ureche.simona@yahoo.com	20
2	Repede	Floresti 2	repede.damaris@yahoo.com	21
3	Ureche	Garii 1082	ureche.elena@yahoo.com	40
9	Bumbu	Valea Caselor 108	bumbu.gabriela@yahoo.com	22
10	Macavei	Granicerilor 6	macavei.al@yahoo.com	25
NULL	NULL	NULL	NULL	NULL

## 4. Implementare



(i) Descrierea claselor și metodelor

(i.1) *Clasa GlientAgeValidator*

Clasa “*GlientAgeValidator*” este responsabilă pentru este responsabilă pentru validarea vârstei unui client. Aceasta conține următoarele metode:

- Metoda `validate(Client client)` este responsabilă pentru validarea vârstei clientului primit ca argument.

(i.2) *Clasa EmailValidator*

Clasa *ProductQuantityValidator* este responsabilă pentru este responsabilă pentru este responsabilă pentru validarea adresei de email unui client. Aceasta conține următoarele metode:

- Metoda `validate(Client client)` este responsabilă pentru validarea adresei de email a clientului primit ca argument..

(i.3) *Clasa ProductQuantityValidator*

Clasa *ProductQuantityValidator* este responsabilă pentru validarea cantității unui produs. Aceasta conține următoarele metode:

- Metoda ``validate(Product product)`` este responsabilă pentru validarea cantității produsului primit ca argument.

#### (i.4.1) Clasa *ClientBLL*:

Clasa *ClientBLL* implementează logica de gestionare a clienților în aplicație. Aceasta conține următoarele metode:

- Metoda ``findAllClients()`` returnează o listă cu toți clienții din baza de date.
- Metoda ``insertClient(Client client)`` inserează un nou client în baza de date, validând în prealabil datele acestuia folosind validatorii din listă.
- Metoda ``updateClient(Client client)`` actualizează un client existent în baza de date, validând în prealabil datele acestuia.
- Metoda ``deleteClient(int id)`` șterge un client din baza de date după id-ul său, aruncând o excepție ``NoSuchElementException`` dacă clientul nu există în baza de date.
- Metodele ``setClientDAO(ClientDAO clientDAO)`` și ``getClientDAO()`` sunt folosite pentru a seta și a obține obiectul ``ClientDAO``, pentru a permite injectarea dependențelor și pentru a accesa funcționalitățile acestuia.

#### (i.4.2) Clasa *OrderBLL*:

Clasa *OrderBLL* se ocupă de logica legată de comenzile clienților. Aceasta conține următoarele metode:

- Metoda ``OrderBLL()`` este constructorul clasei care inițializează obiectul ``OrderDAO``.
- Metoda ``findAllOrders()`` returnează o listă cu toate comenzile din baza de date.
- Metoda ``updateOrder(Orders orders)`` actualizează o comandă existentă în baza de date.
- Metoda ``insertOrder(Orders order)`` inserează o nouă comandă în baza de date.
- Metoda ``deleteOrder(int id)`` șterge o comandă din baza de date după id-ul său, aruncând o excepție ``NoSuchElementException`` dacă comanda nu există în baza de date.
- Metodele ``setOrderDAO(OrderDAO orderDAO)`` și ``getOrderDAO()`` sunt folosite pentru a seta și a obține obiectul ``OrderDAO``, pentru a permite injectarea dependențelor și pentru a accesa funcționalitățile acestuia.

#### (i.5) Clasa *ProductBLL*

Clasa *ProductBLL* se ocupă de logica legată de produse în cadrul aplicației. Aceasta conține următoarele metode:

- Metoda `ProductBLL()` este constructorul clasei care inițializează lista de validatori și obiectul `ProductDAO`.
- Metoda `findProductById(int id)` găsește un produs în baza de date după id-ul său și returnează produsul găsit sau aruncă o excepție `NoSuchElementException` dacă produsul nu există.
- Metoda `findAllProducts()` returnează o listă cu toate produsele din baza de date.
- Metoda `insertProduct(Product product)` inserează un produs în baza de date după ce a fost validat de toți validatorii.
- Metoda `updateProduct(Product product)` actualizează un produs în baza de date după ce a fost validat de toți validatorii.
- Metoda `deleteProduct(int id)` șterge un produs din baza de date după id-ul său, aruncând o excepție `NoSuchElementException` dacă produsul nu există în baza de date.
- Metoda `updateProductQuantity(int productId, int newQuantity)` actualizează cantitatea unui produs în baza de date și apoi actualizează produsul în baza de date după ce a fost validat de toți validatorii.
- Metodele `setProductDAO(ProductDAO productDAO)` și `getProductDAO()` sunt folosite pentru a seta și a obține obiectul `ProductDAO`, pentru a permite injectarea dependențelor și pentru a accesa funcționalitățile acestuia.

#### (i.6) Clasa *ConnectionFactory*

Clasa *ConnectionFactory* gestionează conexiunea la baza de date folosind JDBC. Aceasta conține următoarele metode:

- Conține un constructor privat pentru a preveni instanțierea externă și pentru a asigura o singură instanță a clasei `ConnectionFactory`.
- Metoda `createConnection()` este privată și creează o conexiune la baza de date folosind driverul specificat, URL-ul bazei de date, utilizatorul și parola.
- Metoda `getConnection()` este publică și returnează o instanță a conexiunii la baza de date folosind metoda `createConnection()`.
- Metoda `close(Connection connection)` este publică și închide conexiunea dată ca parametru.
- Metoda `close(Statement statement)` este publică și închide obiectul `Statement` dat ca parametru.
- Metoda `close(ResultSet resultSet)` este publică și închide obiectul `ResultSet` dat ca parametru.
- Utilizează un obiect `Logger` pentru a înregistra mesaje de avertizare în cazul apariției unor erori în timpul operațiilor de conexiune sau închidere.

#### (i.7) Clasa *AbstractDAO*

Clasa *AbstractDAO* furnizează operații generice pentru accesul la date (DAO) folosind JDBC. Aceasta conține următoarele metode:

- Conține un constructor care obține tipul de obiect cu care lucrează clasa DAO folosind reflexia.
- Metoda `createSelectQuery(String field)` este privată și creează o interogare SELECT pentru un anumit câmp specificat.
- Metoda `findById(int id)` caută un obiect în baza de date după ID.
- Metoda `find(String name)` caută obiecte în baza de date după un anumit câmp.
- Metoda `createObjects(ResultSet resultSet)` este privată și creează obiecte pe baza rezultatelor unei interogări SELECT.
- Metoda `findAll()` returnează toate obiectele de tip `T` din baza de date.
- Metoda `insert(T t)` inserează un nou obiect de tip `T` în baza de date.
- Metoda `generateInsertQuery()` este privată și generează interogarea de INSERT pentru inserarea unui nou obiect în baza de date.
- Metoda `setStatementValues(PreparedStatement statement, T t)` este privată și setează valorile declarației pregătite pentru inserarea unui obiect în baza de date.
- Metoda `setGeneratedId(T t, int generatedId)` este privată și setează ID-ul generat al obiectului după inserarea în baza de date.
- Metoda `update(T t)` actualizează un obiect de tip `T` în baza de date.
- Metoda `getIdValue(T t)` este privată și obține valoarea ID-ului obiectului de tip `T`.
- Metoda `delete(String field, int id)` șterge un obiect din baza de date în funcție de un anumit câmp și valoarea acestuia.

#### (i.7) Clasa *Bill*

Clasa *Bill* reprezintă o factură asociată unei comenzi și conține următoarele metode:

- `Bill(int idOrder, int clientId, int productId, int quantity, int totalPrice, String email)`: Constructorul clasei `Bill` care primește ca parametri toate atributele unei facturi.
- Metodele de acces (getteri și setteri) pentru toate atributele clasei, care permit obținerea și setarea valorilor acestora.
- `getId()`: Returnează ID-ul comenzii asociate facturii.
- `getQuantity()`: Returnează cantitatea de produse asociată facturii.
- `getTotalPrice()`: Returnează prețul total al comenzii asociate facturii.
- `getClientId()`: Returnează ID-ul clientului asociat facturii.
- `getProductId()`: Returnează ID-ul produsului asociat facturii.



- ``getEmail()``: Returnează adresa de email asociată facturii.
- ``setId(int id)``: Setează ID-ul comenzii asociate facturii.
- ``setQuantity(int quantity)``: Setează cantitatea de produse asociată facturii.
- ``setTotalPrice(int totalPrice)``: Setează prețul total al comenzii asociate facturii.
- ``setClientId(int clientId)``: Setează ID-ul clientului asociat facturii.
- ``setProductId(int productId)``: Setează ID-ul produsului asociat facturii.

#### (i.8) Clasa Client

Clasa *Client* reprezintă un client și conține următoarele metode:

- ``Client()``: Constructorul fără parametri al clasei ``Client``.
- ``Client(int id, String name, String address, String email, int age)``: Constructorul clasei ``Client`` care primește ca parametri toate atributele unui client.
- ``Client(String name, String address, String email, int age)``: Constructorul clasei ``Client`` care primește ca parametri doar numele, adresa, adresa de email și vârsta clientului.
- Metodele de acces (getteri și setteri) pentru toate atributele clasei, care permit obținerea și setarea valorilor acestora.
- ``getId()``: Returnează ID-ul clientului.
- ``getName()``: Returnează numele clientului.
- ``getAddress()``: Returnează adresa clientului.
- ``getEmail()``: Returnează adresa de email a clientului.
- ``getAge()``: Returnează vârsta clientului.
- ``setId(int id)``: Setează ID-ul clientului.
- ``setName(String name)``: Setează numele clientului.
- ``setAddress(String address)``: Setează adresa clientului.
- ``setEmail(String email)``: Setează adresa de email a clientului.
- ``setAge(int age)``: Setează vârsta clientului.
- ``toString()``: Override pentru metoda ``toString()`` care afișează informații despre client sub formă de șir de caractere.

#### (i.9) Clasa Orders

Clasa *Client* reprezintă o comanda și conține următoarele metode:

- ``Orders()``: Constructorul fără parametri al clasei ``Orders``.
- ``Orders(int id, int clientID, int productID, int quantity, int total_price)``: Constructorul clasei ``Orders`` care primește ca parametri toate atributele unei comenzi.
- ``Orders(int clientID, int productID, int quantity, int total_price)``: Constructorul clasei ``Orders`` care primește ca parametri doar ID-ul clientului, ID-ul produsului, cantitatea și prețul total al comenzii.

- Metodele de acces (getteri și setteri) pentru toate attributele clasei, care permit obținerea și setarea valorilor acestora.
- ``getId()``: Returnează ID-ul comenzii.
- ``getClientID()``: Returnează ID-ul clientului care a plasat comanda.
- ``getProductID()``: Returnează ID-ul produsului comandat.
- ``getQuantity()``: Returnează cantitatea comandată.
- ``getTotal_price()``: Returnează prețul total al comenzii.
- ``setId(int id)``: Setează ID-ul comenzii.
- ``setClientID(int clientID)``: Setează ID-ul clientului care a plasat comanda.
- ``setProductID(int productID)``: Setează ID-ul produsului comandat.
- ``setQuantity(int quantity)``: Setează cantitatea comandată.
- ``setTotal_price(int total_price)``: Setează prețul total al comenzii.

#### (i.10) Clasa *Product*

Clasa *Client* reprezintă un produs și conține următoarele metode:

- ``Product()``: Constructorul fără parametri al clasei ``Product``.
- ``Product(int id, int quantity, int price, String name)``: Constructorul clasei ``Product`` care primește ca parametri toate attributele unui produs.
- ``Product(int quantity, int price, String name)``: Constructorul clasei ``Product`` care primește ca parametri doar cantitatea disponibilă, prețul și numele produsului.
- Metodele de acces (getteri și setteri) pentru toate attributele clasei, care permit obținerea și setarea valorilor acestora.
- ``getId()``: Returnează ID-ul produsului.
- ``getQuantity()``: Returnează cantitatea disponibilă a produsului.
- ``getPrice()``: Returnează prețul produsului.
- ``getName()``: Returnează numele produsului.
- ``setId(int id)``: Setează ID-ul produsului.
- ``setQuantity(int quantity)``: Setează cantitatea disponibilă a produsului.
- ``setPrice(int price)``: Setează prețul produsului.
- ``setName(String name)``: Setează numele produsului.

## 5. Concluzii

Acest proiect m-a ajutat să învăț și să aplic două concepte esențiale în Java: reflection și generice. Prin utilizarea reflectionului, am putut manipula dinamic comportamentul aplicației, iar lucrul cu generice m-a ajutat să scriu cod flexibil și modular. Aceste abilități îmi vor fi de mare folos în proiectele viitoare, permițându-mi să lucrez mai eficient și să implementez soluții mai flexibile și mai extensibile.

## 6. Bibliografie

- Connect to MySQL from a Java application

- <http://theopentutorials.com/tutorials/java/jdbc/jdbc-mysql-create-database-example/>
  - [https://dsrl.eu/courses/pt/materials/PT2024\\_A3\\_S1.pdf](https://dsrl.eu/courses/pt/materials/PT2024_A3_S1.pdf)
- Reflection in Java
  - <https://www.youtube.com/watch?v=bhhMJSKNCQY>
- JAVADOC
  - <https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html>
- Maven
  - <https://www.jetbrains.com/help/idea/maven-support.html>