

**FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE**

**CATEDRA CALCULATOARE**

**ORDER MANAGEMENT**

* **Reflection -**

DOCUMENTAȚIE

Tirlea Maria Cristina

Grupa 30225 | An 2 semestrul 2

Cuprins

**1 . ..... . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ... Obiectiv**

**2... . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Studiul probelemei**

**3.... .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . ......Proiectare**

**3.1.... . . . . . . . . . . . . . . . . . . . . . . . . . . ......Diagrame UML**

**4.... . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . ........Implementare**

**4.1...... . . . . . . . . . . . . . . . .. . . . ... ... . . .. . ... ........... Clase**

**4.2.... . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . .. . ..Metode**

**4.3.... . . . . . . . . . . . . . . . . .. . . .. . . . . . . . .. . . . . . . . ....GUI**

**5… . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . .. .. ...Rezultate**

**6… . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . ...Concluzii**

**7. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . ....Bibliografie**

1. Obiectiv

Scopul acestui proiect este de a crea o aplicatie, cu interfata grafica functionala, care simuleaza un sistem de gestiune a comenzilor unui depozit. Datele despre cienti, produse si comenzi se retin intr-o baza de date care este accesata de catre aplicatie pentru a implementa functinalitatea interfetei.

Ca oricare alt sistem de calcul, si acesta prezinta date de intrare si date de iesire, colectate prin intermediul unei interfete grafice care permite atat introducerea parametrilor de intrare, cat si afisarea datelor de interes din baza de date si modificarile care apar asupra acestora.

Pasii urmati pentru indeplinirea obiectivului:

* Impartirea sarcinilor de implementat in pachete, reprezentand o arhitectura Layered
* Realizarea interfetei prin intermediul careia se colecteaza datale de intrare necesare simularii si se afiseaza informatii din tabelele retinute in baza de date
* Crearea bazei de date, a tabelelor si a legaturilor intre acestea in MySQL Workbench
* Realizarea conexiunii la baza de date prin intermediul API-ului JDBC
* Descompunerea problemei prin definirea claselor model necesare reprezentarii clientilor, produselor si a comenzilor respectand formatul valorilor din tabelele bazei de date
* Implementarea operatiilor asupra datelor din baza de date folosind reflection techniques, tipuri generice si abstractizare
* Realizarea claselor care implementeaza logica corespunzatoarea fiecarei opeartii prin metode specifice fiecarei clase model
* Realizarea functinalitatii componentelor din interfata prin implementarea unei clase de control (Controller)
* Implementarea unei clase principale care manageriaza aplicatia
* Implementarea unei clase utilitare pentru scrierea intr-un fisier de output date despre comenzile efecuate sub forma unei facturi
* Pregatirea rezultatelor pentru a fi prezentate utilizatorului in interfata prin tehnica relflectiei
* Testarea operatiilor in interfata grafica

1. Analiza problemei

# Cerinte functionale

- **Gestiunea stocurilor** este o soluție modernă, extrem de flexibilă, ușor de utilizat, fiind destinată atât societăților comerciale mici, cât și celor mijlocii sau mari din domeniul comerțului. Este soluția ideală pentru automatizarea activității comerciale, pentru managementul stocurilor de produse, evidența vânzărilor și a soldurilor clienților, precum și obținerea raportărilor necesare legate de procesarea comenzilor

- Aplicatia permite retinerea datelor introdu-se prin operatiile puse la dispozitie in interfata, intr-o baza de date creata in aplicatia MySQL Workbench si de asemenea permite manipularea acestor date

- Interfata acestei aplicatii permite utilizatorul poate efectua operatii precum adaugarea / stergerea/ editarea clientilor, adaugarea/ stergerea/ editarea produselor precum si afisarea informatiilor corespunzatoare clientilor si a produselor in tabele

- Utilizatorul poate plasa o comanda pentru produsele aflate in depozit selectand produse si client retinuti in baza de date

- In urma efectuarii unei comenzi se actualizeaza automat stocul produselor comandate si se elibereaza o factura sub forma unui fisier text, care prezinta toate informatiile relevate cu privire la comanda tocmai inregistrata

# Cazuri de utilizare/ Use-cases

**Use-case:** introducerea clientilor in baza de date a depozitului

**Actor principal:** utilizator

**Scenariul principal de succes:**

1. Utilizatorul introduce datele personale ale clientului in campurile text din tab-ul client

2. Utilizatorul selectează butonul “ADD CLIENT”

3. Aplicatia initializeaza campurile din clasa model Client cu valorile preluate de la utilizator

4. In background, se realizeaza operatia de insert in baza de date prin intermediului metodelor specific conexiunii si manipularii datelor

5. Optional, utilizatorul omite acest use-case deoarece clientul care solicita comanda este deja introdus in sistem

**Use-case:** Actualizarea datelor despre un client existent in baza de date

**Actor principal:** utilizator

**Scenariul principal de succes:**

1. Utilizatorul introduce in zona de Edit a tab-ului Client informatiile vechi despre client si valorile cu care doreste sa actualizeze datele deja existente

2. Utilizatorul selectează butonul “EDIT CLIENT”

3. Aplicatia realizeaza operatia de update asupra datelor din tabelul Client al bazei de date

**Use-case:** Afisarea clientilor

**Actor principal:** utilizator

**Scenariul principal de succes:**

1. Utilizatorul selectează butonul “View all clients”

2. Aplicatia foloseste tehnica de reflectie pentru a popula componenta table a interfetei cu informatiile specifice din tabelul Client al bazei de date

**Use-case:** Eliminarea unui client din baza de date

**Actor principal:** utilizator

**Scenariul principal de succes:**

1. Utilizatorul introduce in zona de Add a tab-ului Client informatiile corespunzatoare despre client

2. Utilizatorul selectează butonul “DELETE CLIENT”

3. Aplicatia realizeaza operatia de delete asupra datelor din tabelul Client al bazei de date si sterge din tabela Client randul asociat clientului selectat

**Use-case:** introducerea de produse in baza de date a depozitului

**Actor principal:** utilizator

**Scenariul principal de succes:**

1. Utilizatorul introduce informatiile referitoare la produsul de adaugat in campurile text din tab-ul Product, sectiunea Add

2. Utilizatorul selectează butonul “ADD PRODUCT”

3. Aplicatia initializeaza campurile din clasa model Product cu valorile preluate de la utilizator

4. In background, se realizeaza operatia de insert in baza de date prin intermediului metodelor specifice conexiunii si manipularii datelor

5. Optional, utilizatorul omite acest use-case deoarece produsul solicitat in comanda este deja introdus in sistem

**Use-case:** Actualizarea datelor despre un produs existent in baza de date

**Actor principal:** utilizator

**Scenariul principal de succes:**

1. Utilizatorul introduce in zona de Edit a tab-ului Product informatiile vechi despre produs si valorile cu care doreste sa actualizeze datele deja existente

2. Utilizatorul selectează butonul “EDIT PRODUCT”

3. Aplicatia realizeaza operatia de update asupra datelor din tabelul Product al bazei de date

**Use-case:** Afisarea produselor

**Actor principal:** utilizator

**Scenariul principal de succes:**

1. Utilizatorul selectează butonul “View all products”

2. Aplicatia foloseste tehnica de reflectie pentru a popula componenta table a interfetei cu informatiile specifice din tabelul Product al bazei de date

**Use-case:** Eliminarea unui produs din baza de date

**Actor principal:** utilizator

**Scenariul principal de succes:**

1. Utilizatorul introduce in zona de Add a tab-ului Product informatiile corespunzatoare despre produsul de sters

2. Utilizatorul selectează butonul “DELETE PRODUCT”

3. Aplicatia realizeaza operatia de delete asupra datelor din tabelul Product al bazei de date si sterge din tabela Product randul asociat produsului selectat

**Use-case:** adaugarea unei noi comenzi in baza de date

**Actor principal:** utilizator

**Scenariul principal de succes:**

1. Utilizatorul selectează butonul “Get existing clients/products” pentru a adauga clientii si produsele existente in baza de date in casetele de selectie cu drop-down din interfata (ComboBox-uri)

2. Utilizatorul alege numele produsului, numele clientului si cnp-ul clientului din casetele mentionate la pasul anterior

3. Utilizatorul selectează butonul “ADD ORDER”

3. Aplicatia initializeaza campurile din clasa model Order cu valorile preluate de la utilizator

4. In background, se realizeaza operatia de insert in baza de date prin intermediului metodelor specifice conexiunii si manipularii datelor

**Use-case:** Eliminarea unei comenzi din baza de date

**Actor principal:** utilizator

**Scenariul principal de succes:**

1. Utilizatorul parcurge pasii 1 si 2 mentionati in use case-ul anterior

2. Utilizatorul selectează butonul “DELETE ORDER”

3. Aplicatia realizeaza operatia de delete asupra datelor din tabelul Orders al bazei de date si sterge din tabela Order randul asociat comenzii selectate

**Use-case:** Eliberarea facturii

**Actor principal:** utilizator

**Scenariul principal de succes:**

1. Utilizatorul parcurge pasii mentionati in use case-ul de adaugare a unei comenzi

2. Utilizatorul selectează butonul “RECEIPT”

3. Aplicatia deschide un fisier text care cuprinde informatiile corespunzatoare comenzii tocmai plasate, precum: produsul comandat, cantitatea, pretul total, data si ora efectuarii comenzii

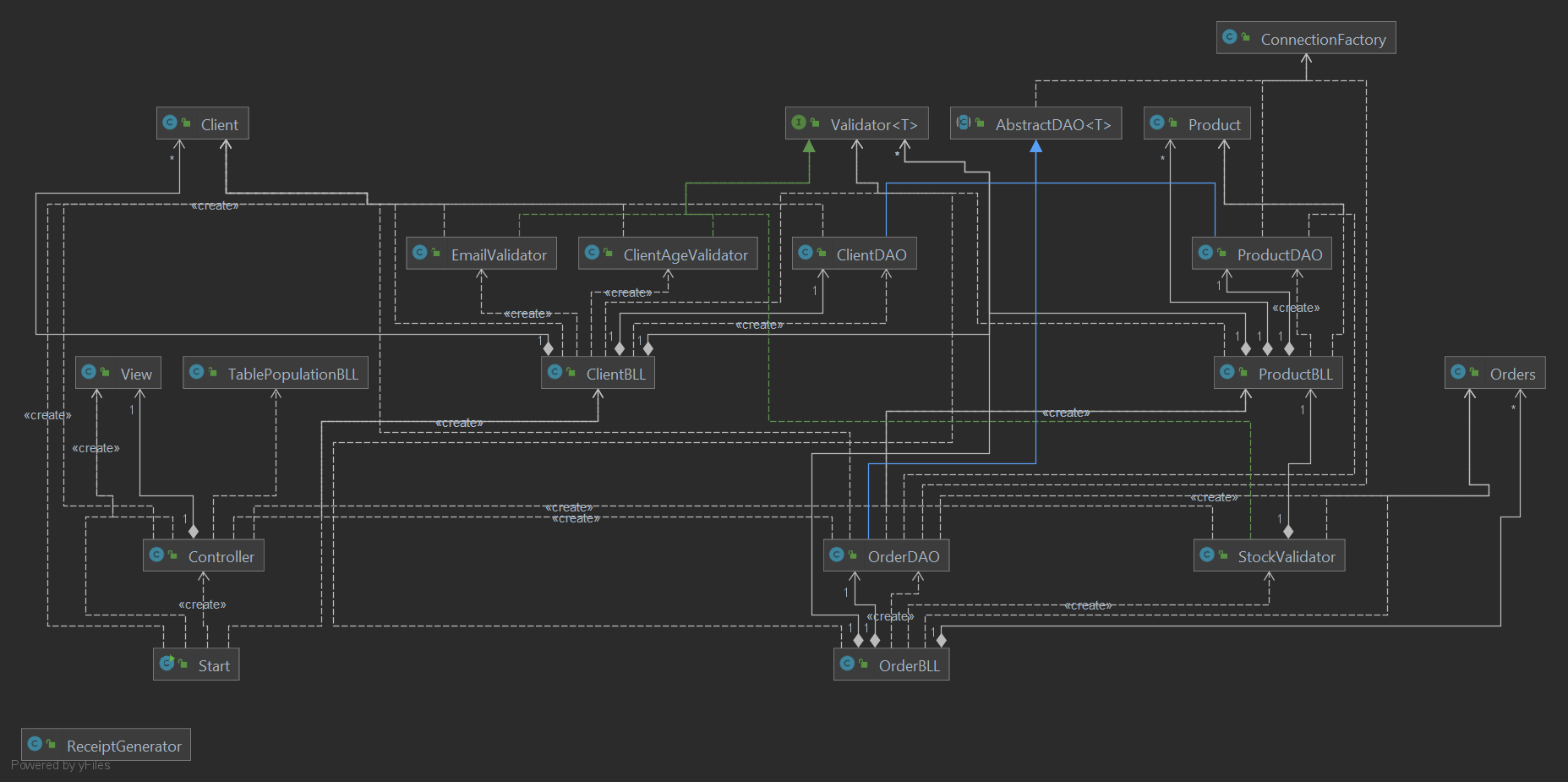
1. Proiectare

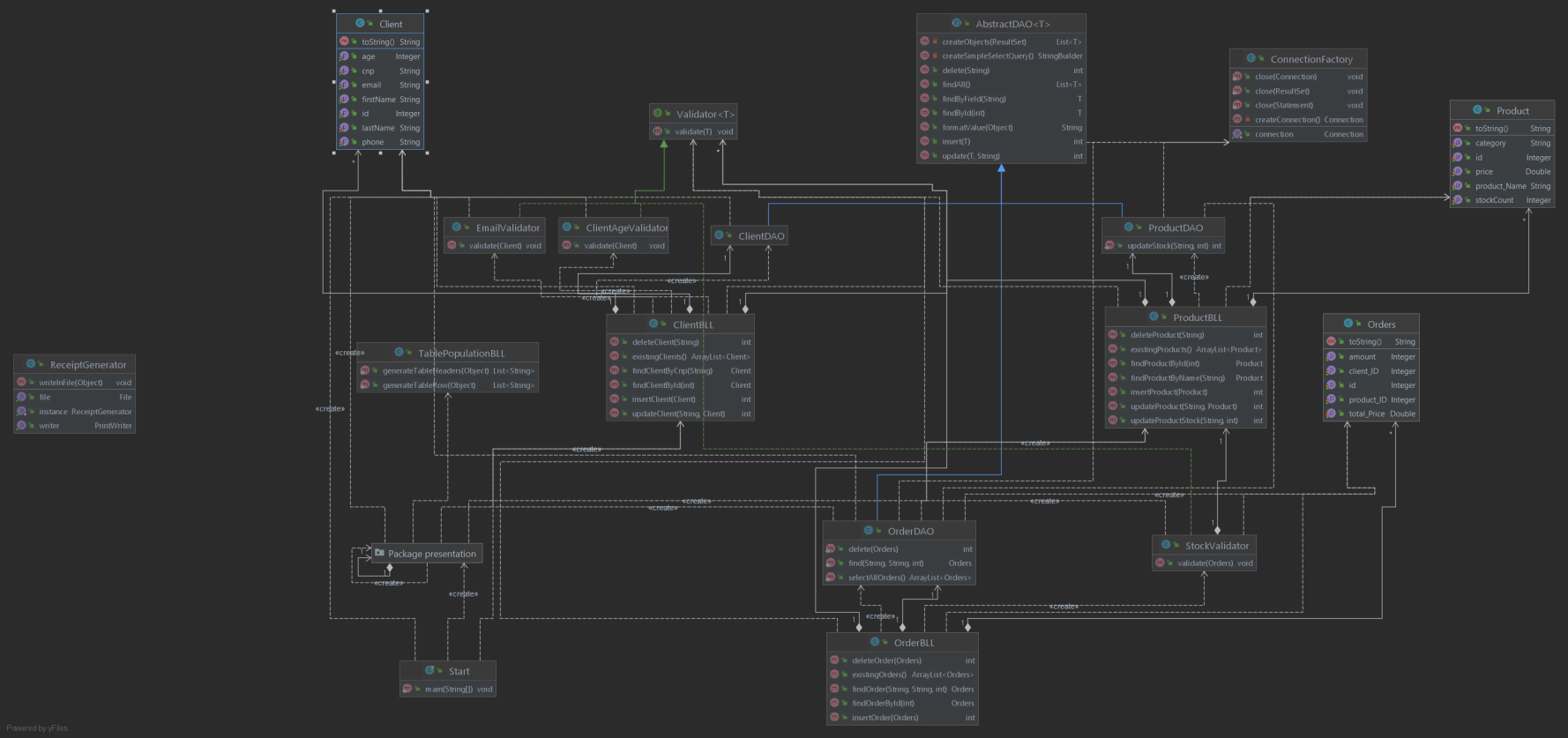
# Diagrama UML

Unified Modeling Language sau UML este un limbaj standard pentru descrierea de modele si specificatii pentru software. UML a fost dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea problemelor in clase, și instanțele acestora (numite și obiecte). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat si dincolo de domeniul IT.

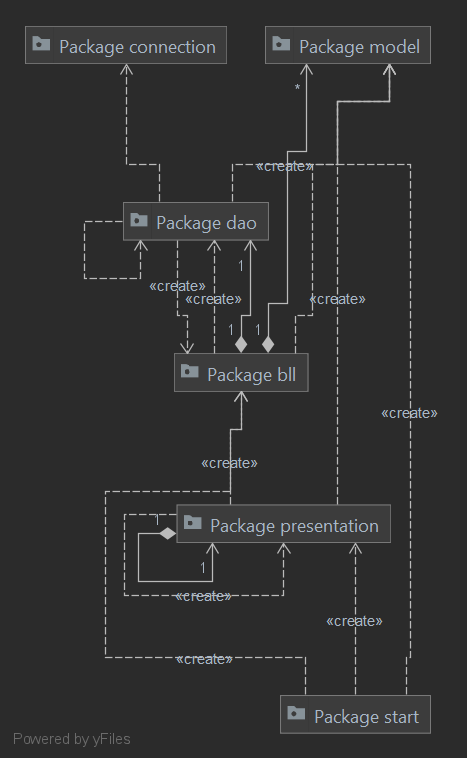
Diagrama de clase oferă o notaţie grafică pentru reprezentarea:

* claselor - entităţi ce au caracteristici comune
* relațiilor - relațiile dintre două sau mai multe clase

**Diagrama UML de clase**:

****

**Diagrama UML de pachete:**



**Proiectarea OOP a aplicatiei:**

Simulatorul de cozi este structurat pe baza pattern-ului architectural Model-View-Controller (MVC) si Layerd Architecture. MVC divizat in 3 zone functionale: modelarea datelor (clasele din pachetul **model**), implementarea interfetei (clasa View pachetul **presentation**), respectiv implementarea legaturii dintre interfata si functinalitatea logica a aplicatiei (clasa Controller din pachetul **presentation**).

Un alt layer al arhitecturii este pachetul **connection**, care contine clase si metode pentru stabilirea conexiunii intre codul java si baza de date creata in MySQL workbench prin intermediul API-ului JDBC.

Pachetul **dao** contine o clasa abstracta care implementeaza operatiile principale asupra bazei de date (insert, delete, update, select, find) prin prisma tehnicii reflectiei si a tipurilor generice. De asememenea contine clase care mostenesc metodele din clasa abstracta si le aplica fiecarei clase model in parte.

Pachetul **validators** contine clase care permit validarea datelor care urmeaza a fi introduse in baza de date, asigurand astfel corectitudinea acestor informatii.

Pachetul **bll** prezinta partea de business logic a aplicatiei si realizeaza legatura intre clasele care modeleaza datele in java (cele din pachetul model) si clasele care obtin informatii de la baza de date (clasele sin pachetele dao si bll).

Pachetul **start** contine clasa principala care ruleaza aplicatia si interfata, precum si clasa ReceiptGenerator care face posibila scrierea facturii.

Alt pattern architectural utilizat in rezolvarea problemei este pattern-ul Singleton. Pattern-ul Singleton este utilizat pentru a restricționa numărul de instanțieri ale unei clase la un singur obiect, deci reprezintă o metodă de a folosi o singură instanță a unui obiect în aplicație. Clasa ReceiptGenerator respecta acest pattern architectural.

Tehnica de reflectie:

Prin ―reflecţie (reflection) se înţelege obţinerea de informaţii despre o clasă sau despre un obiect în faza de execuţie (este ―reflectată starea maşinii virtuale). In plus, se pot crea şi modifica dinamic obiecte în faza de execuţie. Reflecţia este asigurată în principal de clasa numită Class, dar şi de alte clase din pachetul java.lang.reflect:

* Constructor
* Method
* Field
* Annotation
* Class

Fiecare dintre aceste clase abstractizează conceptul corespunzător numelui clasei. Implementarea internă a acestui mecanism este diferită de la un JDK la altul, dar în mare este bazată pe metode native.

Clasa Class conţine metode care au ca rezultat numele clasei, tipul clasei (clasă sau interfaţă sau vector), tipul superclasei, clasa externă, interfeţe implementate, tipul obiectelor declarate în clasă, numele câmpurilor (variabilelor clasei), numele metodelor clasei, formele funcţiilor constructor s.a.

Metoda getName() are ca rezultat un şir ce reprezintă numele clasei al cărui tip este conţinut într-un obiect Class.

Prin reflecţie un asamblor de componente JavaBeans dintr-un mediu vizual poate să determine proprietăţile si metodele proprii unor obiecte, să modifice proprietăţile acestor obiecte şi să genereze apeluri de metode între obiecte. In rezumat, reflecţia permite operaţii cu clase şi cu obiecte necunoscute la scrierea programului, dar care pot fi determinate dinamic, în cursul execuţiei.

Dacă se apelează metode polimorfice pentru obiecte de tip necunoscut, atunci se va apela automat varianta definită în clasa respectivă, chiar dacă programatorul (si nici compilatorul) nu ştie care este tipul exact al obiectului.

Am utilizat aceasta tehnica pentru a implementa operatiile uzuale asupra tabelelor din baza de date, pentru a genera header-ele tabelelor din interfata si de asemenea pentru a obtine valorile de pe fiecare rand din tabelele din baza de date, valori cu care am populat ulterior tabelele din interfata.

1. Implementare

# . Clase

**Clasa ClientAgeValidator**

Clasa utilizată pentru verificarea limitei de vârstă a fiecărui client inserat în sistemul de baze de date.

**Clasa EmailValidator**

Clasa utilizată pentru verificarea respectarea sablonului specific de e-mail al cariabilei email a fiecărui client inserat în sistemul de baze de date.

**Clasa StockValidator**

Clasa utilizată pentru verificarea existentei de stoc suficient pentru produsul pentru care se plaseaza o comandă.

**Clasa ClientBLL**

Clasă care implementează logica pentru clasa model Client și metodele pentru accesarea DB printr-o instanță a clasei ClientDAO

**Clasa OrderBLL**

Clasa de implementare a logicii pentru clasa model Orders și a metodelor pentru accesarea DB printr-o instanță a OrderDAO

**Clasa ProductBLL**

Clasă care implementează logica pentru clasa modelului Produs și metodele de accesare a bazei de date printr-o instanță a clasei ProductDAO

**Clasa TablePopulationBLL**

Clasă care implementează metode pentru popularea componentelor JTable în UI, utilizând tehnica de reflective

**Clasa ConnectionFactory**

Clasa care oferă conexiunea la baza de date

**Clasa AbstractDAO**

Clasa abstracta care folosește generice și tehnica de reflecție pentru a defini operațiunile comune pentru accesarea unui tabel: Insert, Update, Delete, FindById, FindAll

**Clasa ClientDAO**

Clasa care moștenește metodele din clasa abstractă AbstractDAO.java și oferă utilizarea acestora pentru obiectele de tip Client

**Clasa OrderDAO**

Clasa care moștenește metodele din clasa abstractă AbstractDAO.java și oferă utilizarea acestora pentru obiectele de tip Orders

**Clasa ProductDAO**

Clasa care moștenește metodele din clasa abstractă AbstractDAO.java și oferă utilizarea acestora pentru obiectele de tip Product si implementeaza si alte metode specifice

**Clasa Client**

Clasa care modelează tabelul Client din baza de date

**Clasa Product**

Clasa care modelează tabelul Product din baza de date

**Clasa Orders**

Clasa care modelează tabelul Orders din baza de date

**Clasa ReceiptGenerator**

Clasa care implementează un model de proiectare Singleton pentru a scrie într-un fișier informațiile de facturare pentru o comandă care a fost plasată

**Clasa Start**

Clasa principal care ruleaza aplicatia

**Clasa View**

* Incapsuleaza componentele care alcatuiesc interfata grafica (vederea).

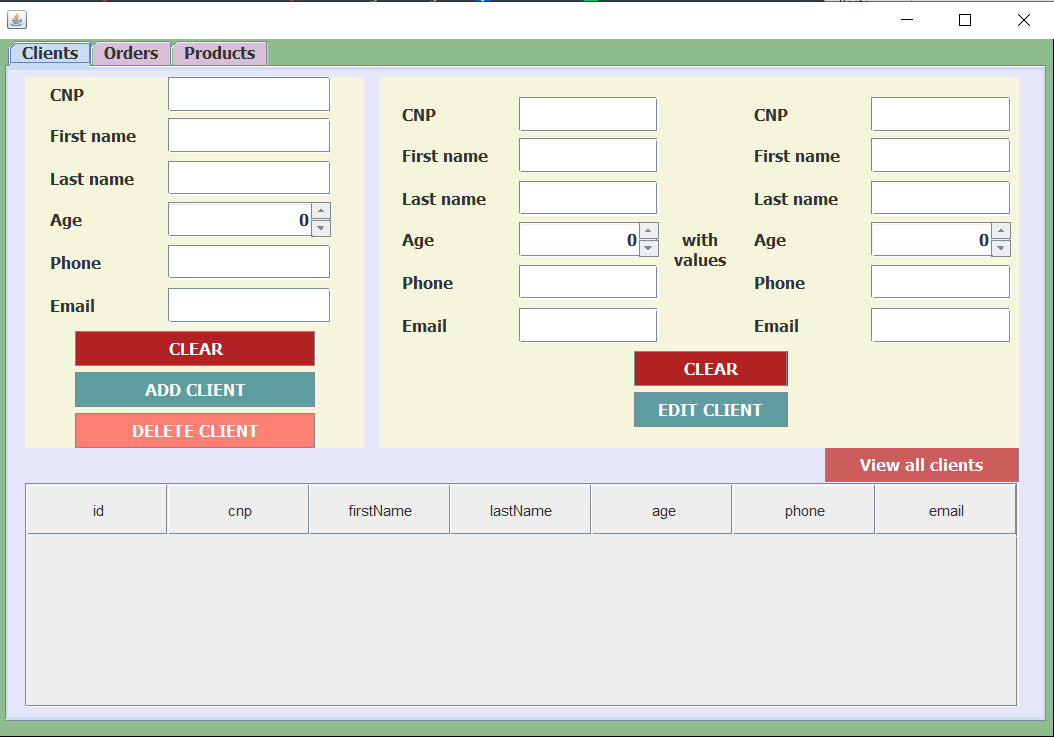
**Clasa Controller**

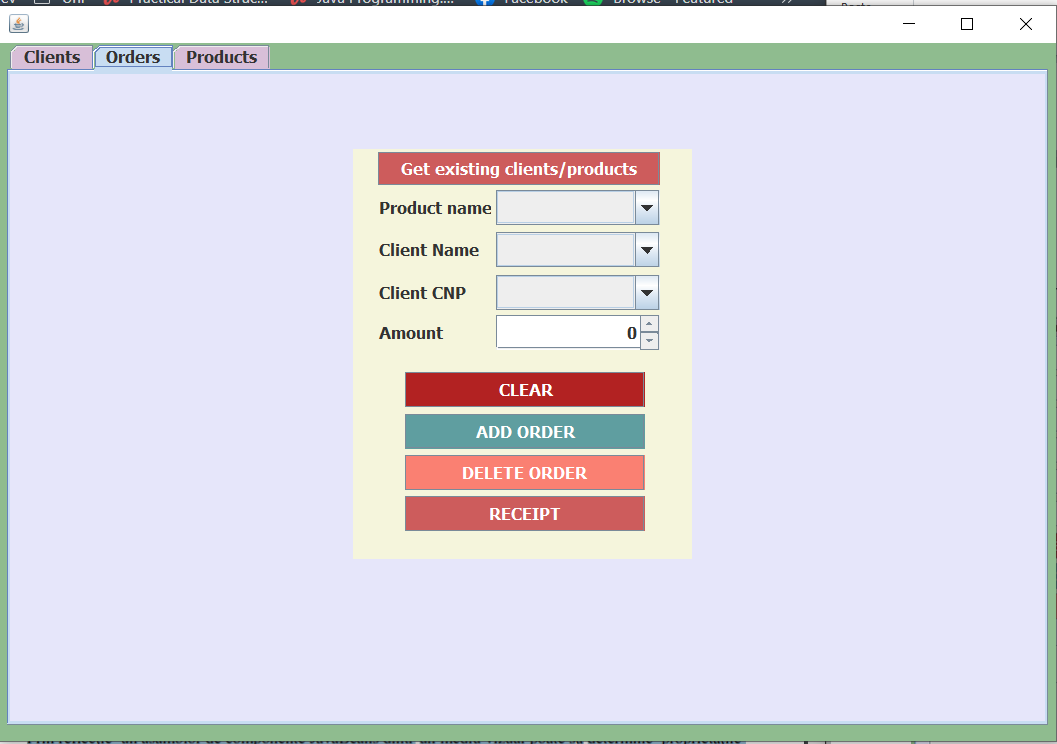
* Clasa Controller primeste datele introduse de utilizator in interfata ca evenimente care denotă mișcarea mouse-ului, activarea butoanele mouse-ului sau de la tastatura.
* Permite interactiunea intre utilizator si sistemul de gestionare a comenzilor depozitului.
* Face lagatura intre interfata si functionalitatea oferita de componentele de modelare.
  1. **. Metode**

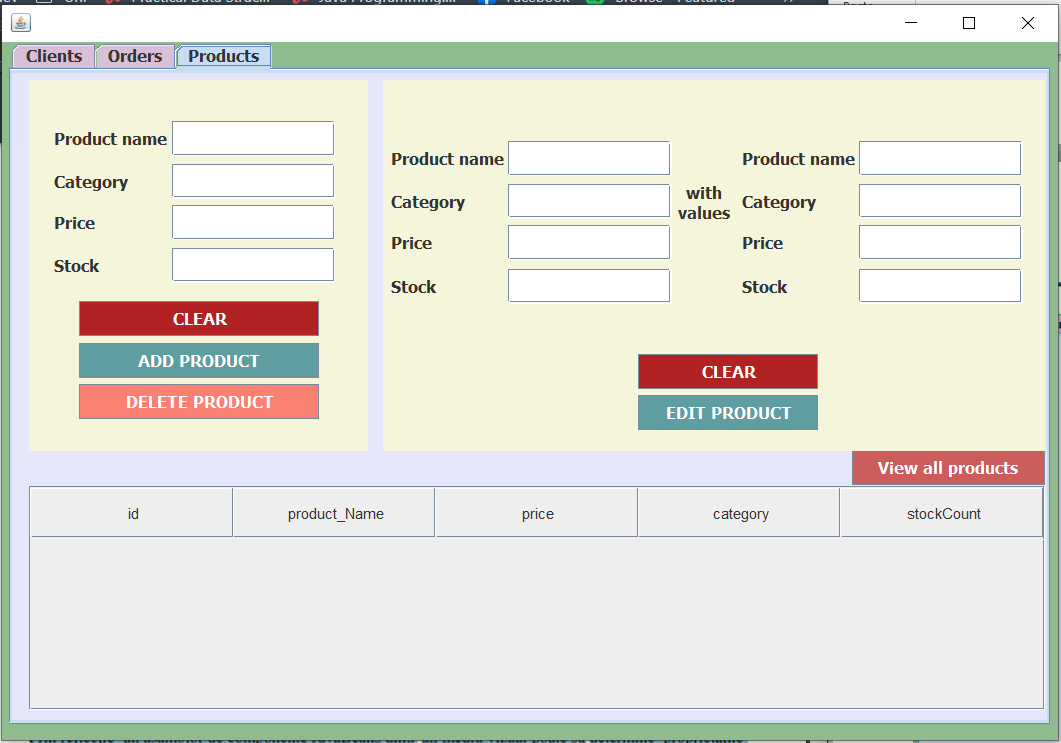
**Metodele fiecarei clase sunt prezentate in detaliu, impreuna cu descrierea parametrilor si a valorilor returnate in documentul javadoc**

**4.3. GUI**

- Interfata implementata in clasa View contine elemente din pachetul java.Swing si cod pentru layout-ul acestor elemente :







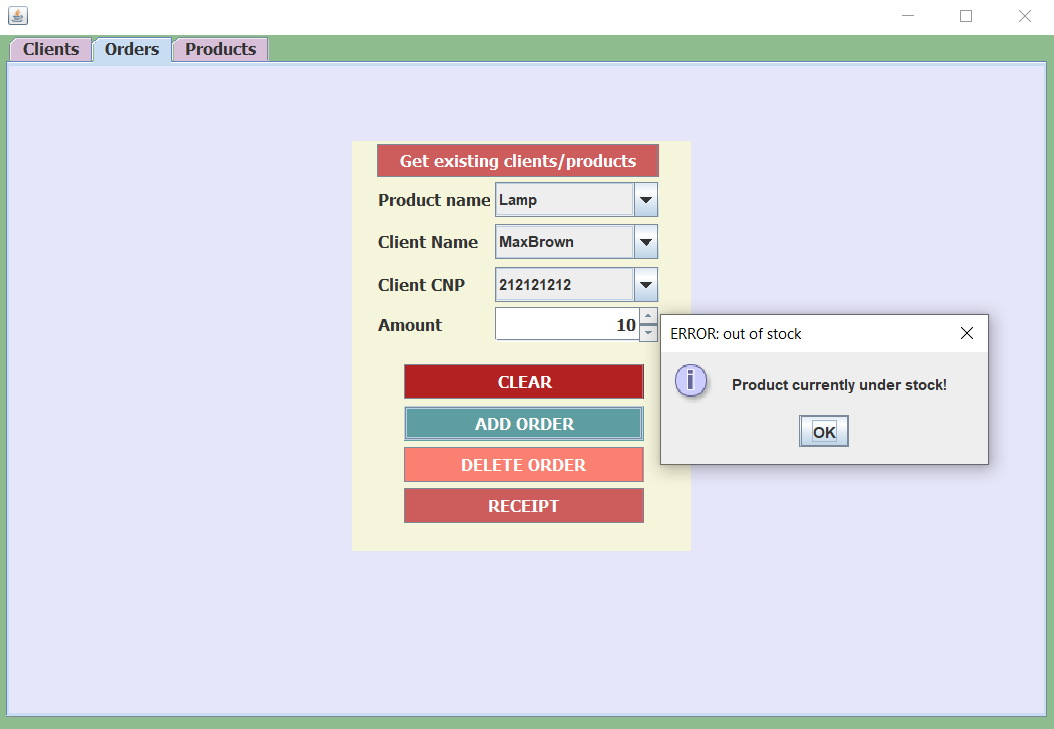
Pentru fiecare din cele trei modele de date (Produs, Client, Comanda) am creat in interfata cate un tab cu componentele corespunzatoare actiunilor de realizat asupra fiecarui tip de data.

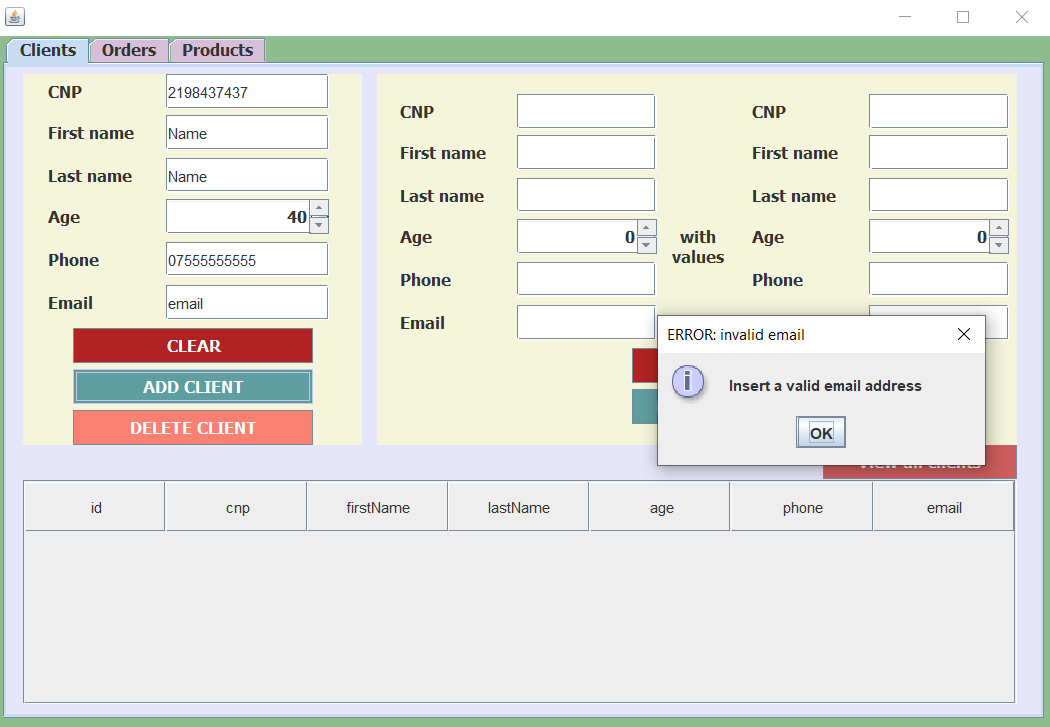
Comunicarea intre interfata/ utiliator si codul de funcionalitate din spate este realizata cu ajutorul clasei Controller si al claselor interne definite in clasa Controller care implementeaza interfete precum ActionListener, care asculta evenimente asupra interfetei precum mișcarea mouse-ului, activarea butoanele mouse-ului sau activarea tastelor de la tastatura. Evenimentele sunt ulterior traduse în cereri de servicii, care sunt trimise fie la model-ul de datesi la clasele de businesslogic , fie la View.

1. Testare

Pentru testarea functionarii dorite a aplicatiei am luat fiecare use-case in parte si am testat fiecare functionalitate pusa la dispozitie in interfata grafica.

Totodata am testat cazurile in care se introduce valori care nu verifica testele din clasele de vaidare, astfel ca aplicatia a deschis in aceste cazuri ferestre de tip pop-up cu mesajele de eroare corespunzatoare.





1. Concluzii

In concluzie, consider ca acest proiect mi-a aprofundat cunostiintele de programare in limbajul Java si de implementare a paradigmelor OOP. Problema a cuprins toate etapele importante ale dezvoltarii unei aplicatii cu interfata grafica respectand standardul OOP. Realizarea acestui proiect m-a ajutat sa inteleg tehnica reflection prin aplicare practica a acesteia si a conceptelor de abstractizare si structurare a aplicatiilor in mai multe straturi (pachete).

In completarea acestei documentatii, am realizat un javadoc care prezinta in detaliu componentele care alcatuiesc acesta aplicatie si functionalitatea fiecaruia.

1. Bibliografie

* FUNDAMENTAL PROGRAMMING TECHNIQUES ASSIGNMENT 3 – SUPPORT PRESENTATION
* Connect to MySQL from a Java application

<https://www.baeldung.com/java-jdbc>

<http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/>

* Layered architectures

<https://dzone.com/articles/layers-standard-enterprise>

* Reflection in Java

<http://tutorials.jenkov.com/java-reflection/index.html>

* JAVADOC

<https://www.baeldung.com/javadoc>

* SQL dump file generation

<https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>

◦ Simple layered project: <https://gitlab.com/utcn_dsrl/pt-layered-architecture>

◦ Reflection example: <https://gitlab.com/utcn_dsrl/pt-reflection-example>