

**UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA  
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**

**Tehnici de programare fundamentale  
Tema 3**

# **Management pentru comenzi**

**- Documentație -**

*Realizat de:*  
Ducai David Alex – grupa 30223

## *Cuprins*

1. Obiectivele temei .....	3
2. Analiza problemei.....	4
3. Proiectarea temei .....	7
4. Implementarea temei .....	11
5. Testarea temei și rezultate .....	16
6. Concluzii și dezvoltări ulterioare.....	18
7. Webografie .....	19

## 1. Obiectivele temei

**Obiectivul principal** al temei este realizarea unei aplicații conectate la o bază de date relațională, care reține date despre clienții, produsele și comenzile unui depozit sau ale unei afaceri. Aplicația trebuie să aibă o interfață grafică, care va permite introducerea, modificarea, ștergerea și afișarea înregistrărilor din tabelele bazei de date conectate.

**Obiectivele secundare** ale temei sunt următoarele:

- Analizarea problemei și identificarea resurselor necesare realizării temei propuse – *capitolul 2*
- Realizarea design-ului aplicației – *capitolele 3 și 4*
- Implementarea aplicației – *capitolul 4*
- Testarea aplicației – *capitolul 5*

## 2. Analiza problemei

### 2.1. Cerințe funcționale

Aplicația trebuie să permită utilizatorului să introducă, să actualizeze, să ștergă și să afișeze clienți și produse înregistrate în baza de date, și să selecteze un produs și un client pentru a adăuga o comandă. În spatele interfeței, calculatorul trebuie să acceseze baza de date și să o actualizeze în timp real conform acțiunilor utilizatorului în aplicație.

De asemenea, aplicația trebuie să genereze câte o factură pentru fiecare comandă validă introdusă. Factura va conține date despre clientul care a comandat, produsul comandat, prețul final, cantitatea comandată, etc.

### 2.2. Cerințe non-funcționale

Aplicația trebuie să vină însoțită de o interfață intuitivă, ușor de folosit de către utilizator. Aplicația trebuie să valideze datele de intrare introduse de către utilizator și să afișeze diferite mesaje care să arate succesul sau eșecul manipulării bazei de date.

### 2.3 Cazuri de utilizare

Pentru toate cazurile de utilizare prezentate, *actorul principal este utilizatorul* aplicației, care poate fi un angajat al afacerii sau un administrator. Anumite operații au cazuri de utilizare similare și vor fi prezentate simultan, pentru simplificare.

#### 1. *Caz de utilizare: introducerea unui client / produs în baza de date*

**Scenariu principal** (operație realizată cu succes):

1. Utilizatorul apasă pe butonul care deschide secțiunea pentru clienți / produse (“Accesare clienți” / “Accesare produse”)
2. Utilizatorul selectează butonul care deschide secțiunea pentru adăugarea unei noi înregistrări (“Adăugare”)
3. Utilizatorul introduce datele necesare în câmpurile afișate
4. Utilizatorul apasă butonul de adăugare în baza de date (“Adăugare client” / “Adăugare produs”)
5. Utilizatorul primește un mesaj care indică reușita realizării operației
6. Utilizatorul poate vedea rezultatul inserării (*Cazul de utilizare 2.*)

**Scenariu secundar** (Email greșit - *client*):

- Utilizatorul nu a introdus o adresă de email validă
- Se deschide o nouă fereastră de atenționare pentru utilizator
- Se revine în scenariul principal la pasul 3.

**Scenariu secundar** (Stoc negativ - *produs*):

- Utilizatorul nu a introdus o valoare a stocului mai mare sau egală cu 0 pentru un produs
- Se deschide o nouă fereastră de atenționare pentru utilizator
- Se revine în scenariul principal la pasul 3.

**Scenariu secundar** (Conexiune nereușită cu baza de date):

- Utilizatorul introduce date corecte în interfață (sau nu – *depinde de cazul de utilizare*) și primește un mesaj de atenționare care indică insuccesul realizării adăugării din diferite motive
- Se închide aplicația
- Se verifică serverul / conexiunea la baza de date
- Se revine în scenariul principal la pasul 1.

**Acest scenariu secundar se aplică la toate cazurile de utilizare!**

#### 2. *Caz de utilizare: afișarea clienților / produselor din baza de date*

**Scenariu principal** (operație realizată cu succes):

1. Utilizatorul apasă pe butonul care deschide secțiunea pentru clienți / produse (“Accesare clienți” / “Accesare produse”)
2. Utilizatorul selectează butonul care deschide tabelul cu înregistrările din baza de date corespunzătoare (“Listare”)

### 3. *Caz de utilizare: actualizarea unui client / produs*

#### **Scenariu principal** (operație realizată cu succes):

1. Utilizatorul apasă pe butonul care deschide secțiunea pentru clienți / produse (“Accesare clienți” / “Accesare produse”)
2. Utilizatorul selectează butonul care deschide tabelul cu înregistrările din baza de date corespunzătoare (“Listare”)
3. Utilizatorul selectează rândul din tabel, pe care dorește să îl modifice
4. Utilizatorul selectează butonul care deschide secțiunea pentru actualizarea unei înregistrări (“Actualizare”)
5. Se afișează secțiunea de actualizare, în care câmpurile sunt completate cu datele actuale de pe rândul selectat
6. Utilizatorul poate modifica datele
7. Utilizatorul apasă butonul de actualizare (“Actualizare client” / “Actualizare produs”)
8. Utilizatorul primește un mesaj care indică reușita realizării operației
9. Utilizatorul poate vedea rezultatele (*Cazul de utilizare 2.*)

#### **Scenariu secundar** (Email greșit - *client*):

- Scenariu asemănător cu primul scenariu secundar din *Cazul de utilizare 1*
- Se revine în acest scenariu principal la pasul 6.

#### **Scenariu secundar** (Stoc negativ - *produs*):

- Scenariu asemănător cu al doilea scenariu secundar din *Cazul de utilizare 1*
- Se revine în acest scenariu principal la pasul 6.

### 4. *Caz de utilizare: ștergerea unui client / produs*

#### **Scenariu principal** (operație realizată cu succes):

1. Utilizatorul apasă pe butonul care deschide secțiunea pentru clienți / produse (“Accesare clienți” / “Accesare produse”)
2. Utilizatorul selectează butonul care deschide tabelul cu înregistrările din baza de date corespunzătoare (“Listare”)
3. Utilizatorul selectează rândul din tabel, pe care dorește să îl șteargă
4. Utilizatorul apasă pe butonul pentru ștergere (“Ștergere”)
5. Utilizatorul primește un mesaj care indică reușita realizării operației
6. Utilizatorul poate vedea rezultatele

### 5. *Caz de utilizare: adăugarea unei comenzi*

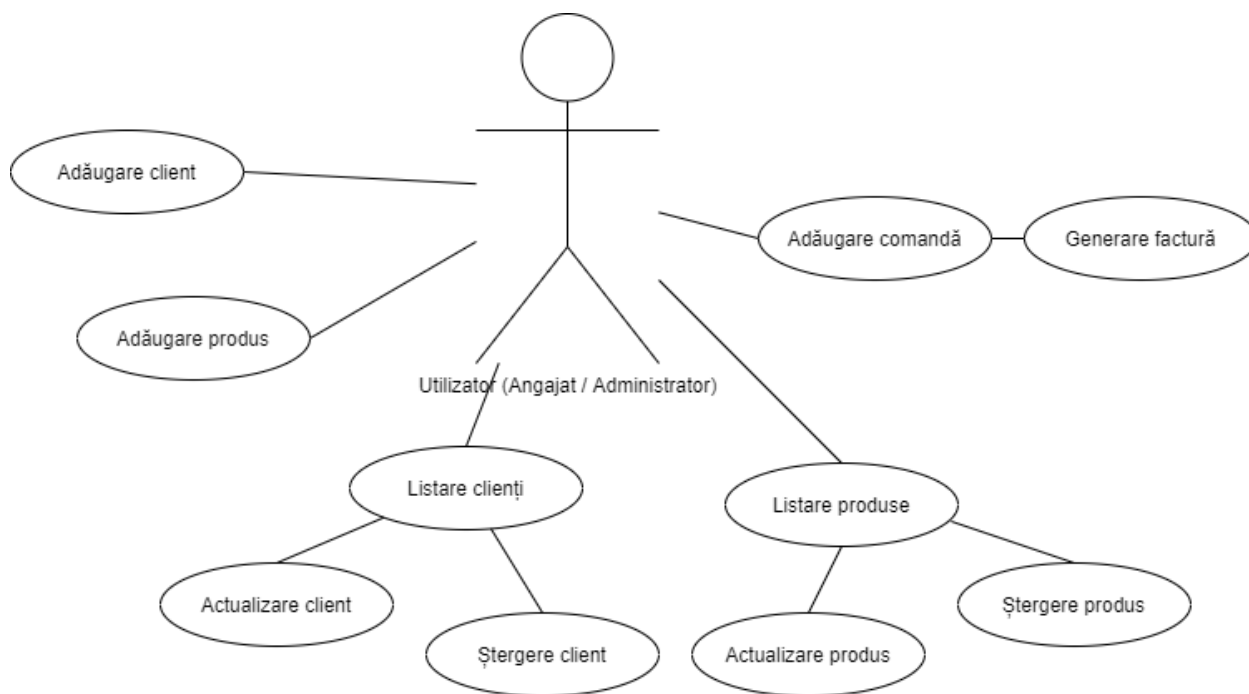
#### **Scenariu principal** (operație realizată cu succes):

1. Utilizatorul apasă butonul de deschidere a secțiunii pentru comenzi (“Accesare comenzi”)
2. Se deschide o nouă fereastră cu tabela clienților
3. Utilizatorul selectează din tabel clientul dorit și apasă butonul de continuare
4. Se deschide o nouă secțiune cu tabelul produselor în stoc
5. Utilizatorul selectează din tabel produsul dorit și apasă butonul de continuare
6. Se deschide ultima secțiune, unde utilizatorul introduce cantitatea comandată din produsul selectat
7. Utilizatorul apasă butonul de finalizare
8. Se introduce în baza de date comanda și se generează factura
9. Se afișează mesaje care indică succesul operațiilor
10. Se închide automat fereastra

#### **Scenariu secundar** (Cantitate nevalidă):

- Utilizatorul nu a introdus o cantitate validă (mai mare decât 0)
- Se deschide o nouă fereastră de atenționare pentru utilizator
- Se revine în scenariul principal la pasul 6.

- Use Case Diagram:



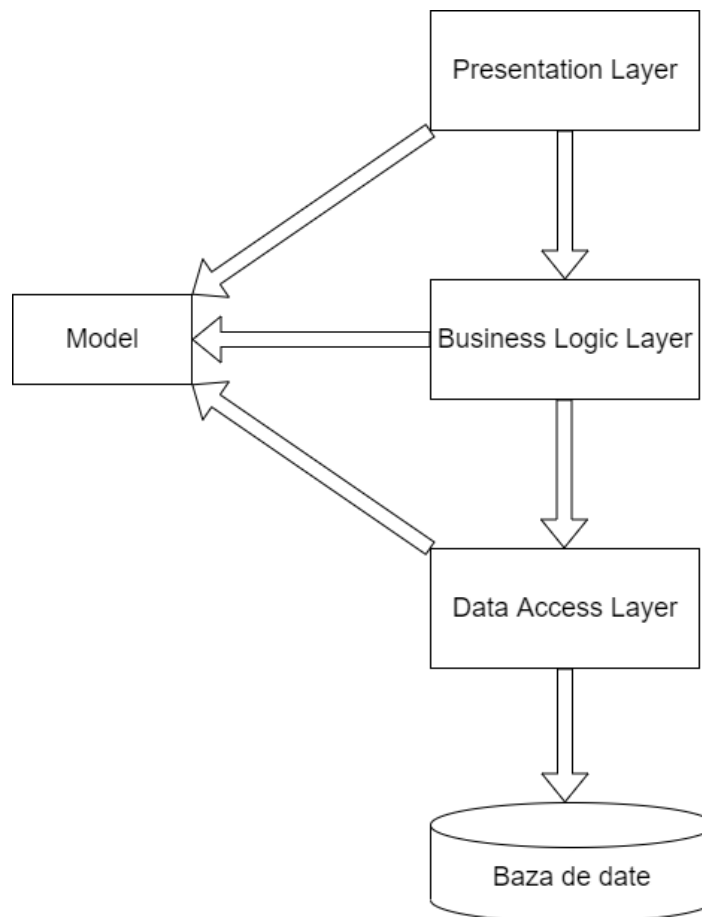
### 3. Proiectarea temei

Pentru realizarea acestei aplicații am ales modelul arhitectural *Three-Tier Architecture*. Acest model arhitectural desparte aplicația în trei etaje / straturi (layere):

- **Etajul de prezentare** (Presentation Layer) – conține clasele care definesc interfața grafică a aplicației (view, controller, etc.)
- **Etajul de logică a afacerii** (Business Logic Layer) – conține clasele care încapsulează logica aplicației (validatori pentru date de intrare, etc.)
- **Etajul de acces la date** (Data Access Layer) – conține clasele care rețin interogările bazei de date și accesul la aceasta

Toate aceste etaje au acces la un pachet **Model**, care conține clasele mapate în baza de date (în cazul nostru, clasele pentru client, produs și comandă).

Fiecare etaj are funcționalități specifice și apelează funcții din etajul de dedesubt:



În baza de date relațională se vor crea trei tabele:

- **client** – *attribute*: id, nume, email, adresă, oraș, țară
- **produs** – *attribute*: id, nume, preț unitar, stoc
- **comandă** – *attribute*: id, id client, id produs, prețul final, cantitatea comandată și data comenzii

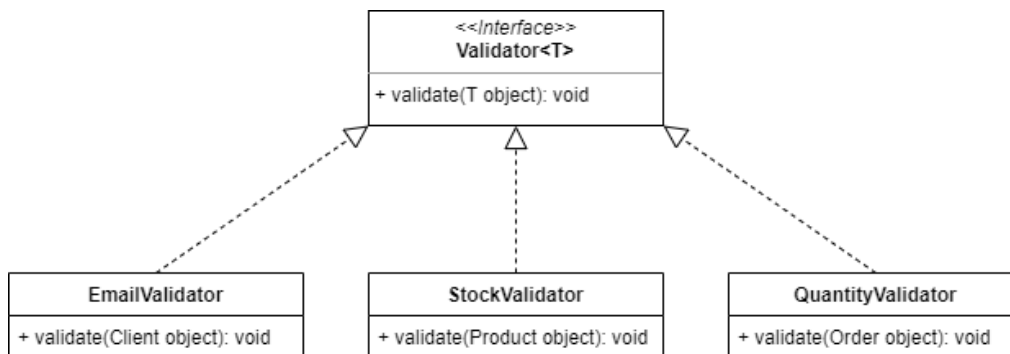
Primul pachet care va fi implementat este pachetul pentru *modelul de date*, deoarece toate etajele vor utiliza clase din acesta. Se vor crea clasele **client**, **produs** și **comandă** cu atributele prezentate anterior. Clasele vor conține constructori și metode pentru accesare și modificarea atributelor. Aceste clase vor fi utilizate pentru: salvarea atributelor unei înregistrări din baza de date, prezentare în interfață, validare, facturare, etc.

Pe lângă etajele menționate anterior, va mai fi implementat un pachet pentru *conexiunea cu baza de date*. Acest pachet va conține o singură clasă, **ConnectionFactory**, care va avea ca atribute: o conexiune cu baza de date, un link către aceasta, numele de utilizator, parola, etc. Clasa va avea metode pentru realizarea unei conexiuni și pentru închiderea în siguranță a acesteia. Această clasă va fi folosită numai în etajul de acces la baza de date.

Următorul pachet care va fi implementat va fi pachetul pentru etajul **Data Access**. Pentru fiecare tip de obiect din pachetul Model va trebui creat câte un obiect pentru accesul la baza de date, **Data Acces Object (DAO)**. Fiecare obiect de acest tip va conține metode pentru manipularea bazei de date (adăugare, actualizare, ștergere, selectare). Pentru eficiență, vom evita descrierea manuală a metodelor pentru fiecare obiect și vom crea o clasă generică **AbstractDAO**. Fiecare obiect DAO pentru client, produs și comandă va extinde această clasă. În consecință, vor trebui scrise doar metodele generice pentru AbstractDAO, iar celelalte obiecte DAO vor moșteni prin tehnica *Reflection* aceste metode pentru modelul pe care îl utilizează.

Urmează pachetul pentru etajul **Business Logic**. Acest pachet va conține clase pentru logica fiecărui tabel din baza de date (*ClientBL*, *ProductBL*, *OrderBL*) și pentru validare. În acest pachet va fi implementată interfața **Validator** cu un tip generic, care va avea o singură metodă pentru validare. Clasele pentru validare vor implementa această interfață pentru modelul pe care îl utilizează. Fiecare clasă validator va verifica, practic, dacă sunt încălcate anumite reguli. În caz afirmativ, clasele vor arunca câte o excepție cu un mesaj care va fi afișat în aplicație. Validatorii pentru fiecare model vor fi adăugați într-o *listă de validatori*, atribut al clasei BL corespunzătoare. Fiecare clasă BL va avea metode care vor valida obiectele introduse/actualizate și vor manipula baza de date cu ajutorul *obiectului DAO* corespunzător modelului utilizat. Obiectul DAO este și el un atribut al clasei BL.

Pentru aplicație, vom avea nevoie de validatori, de exemplu, pentru: email (client), stoc (produs) și cantitate (comandă).



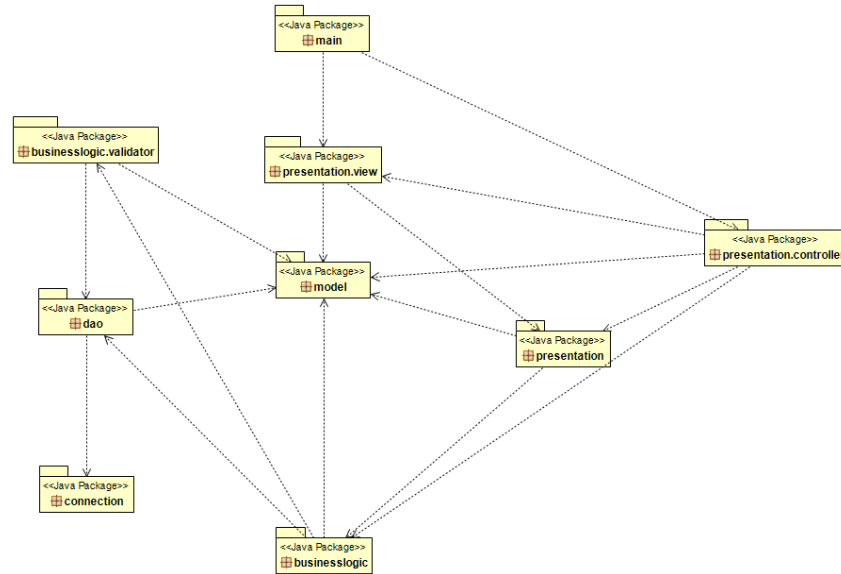
Următorul pachet implementat va fi pachetul pentru **prezentare**. Va conține: clasele view și controller pentru fiecare model, o clasă **Tabel** cu tip generic pentru afișarea rezultatelor din baza de date, și o clasă **Bill** pentru generarea facturii la fiecare comandă.

Ultimul pachet, **main**, conține doar clasa pentru execuția aplicației.

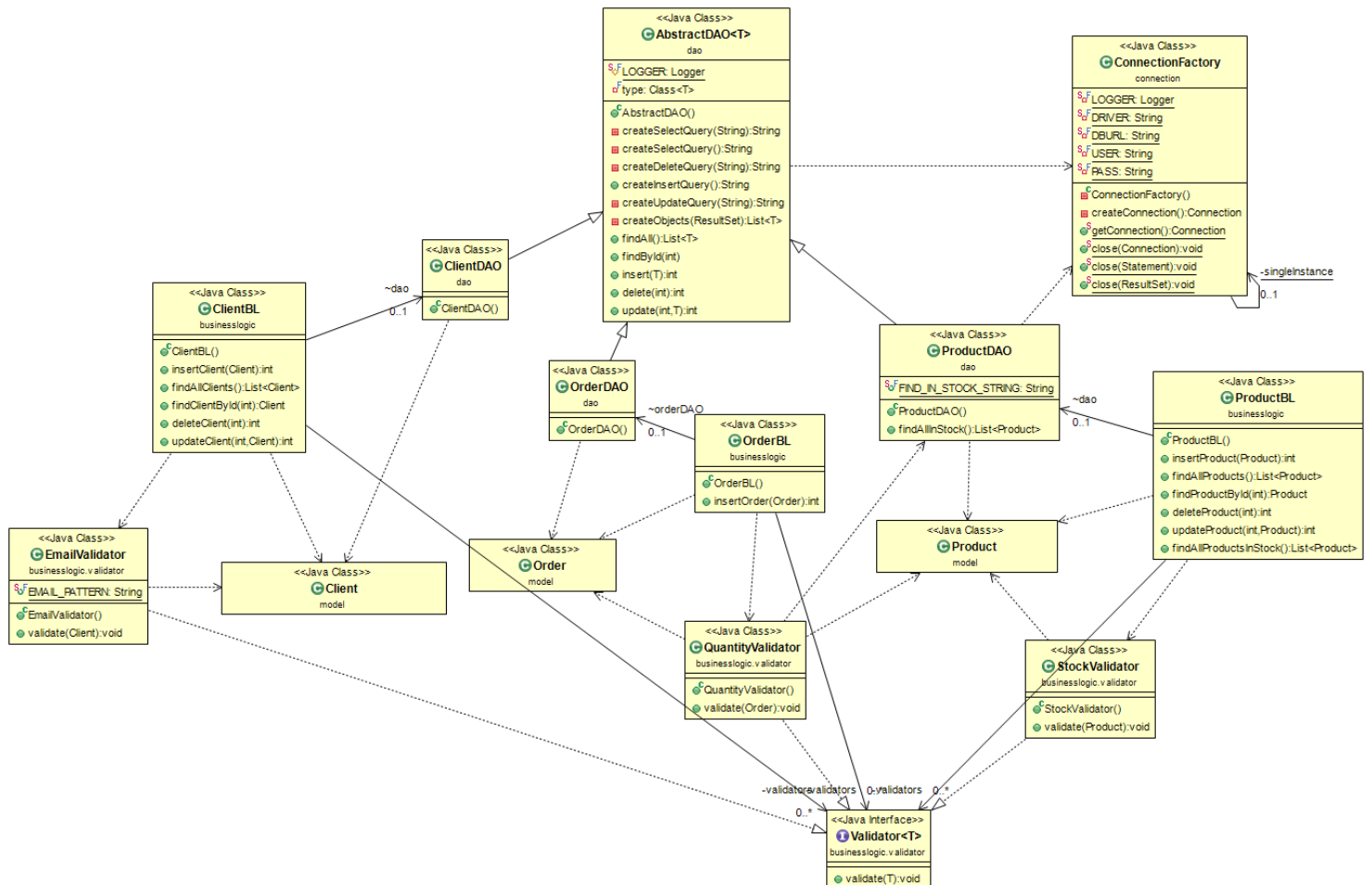


Diagrame UML:

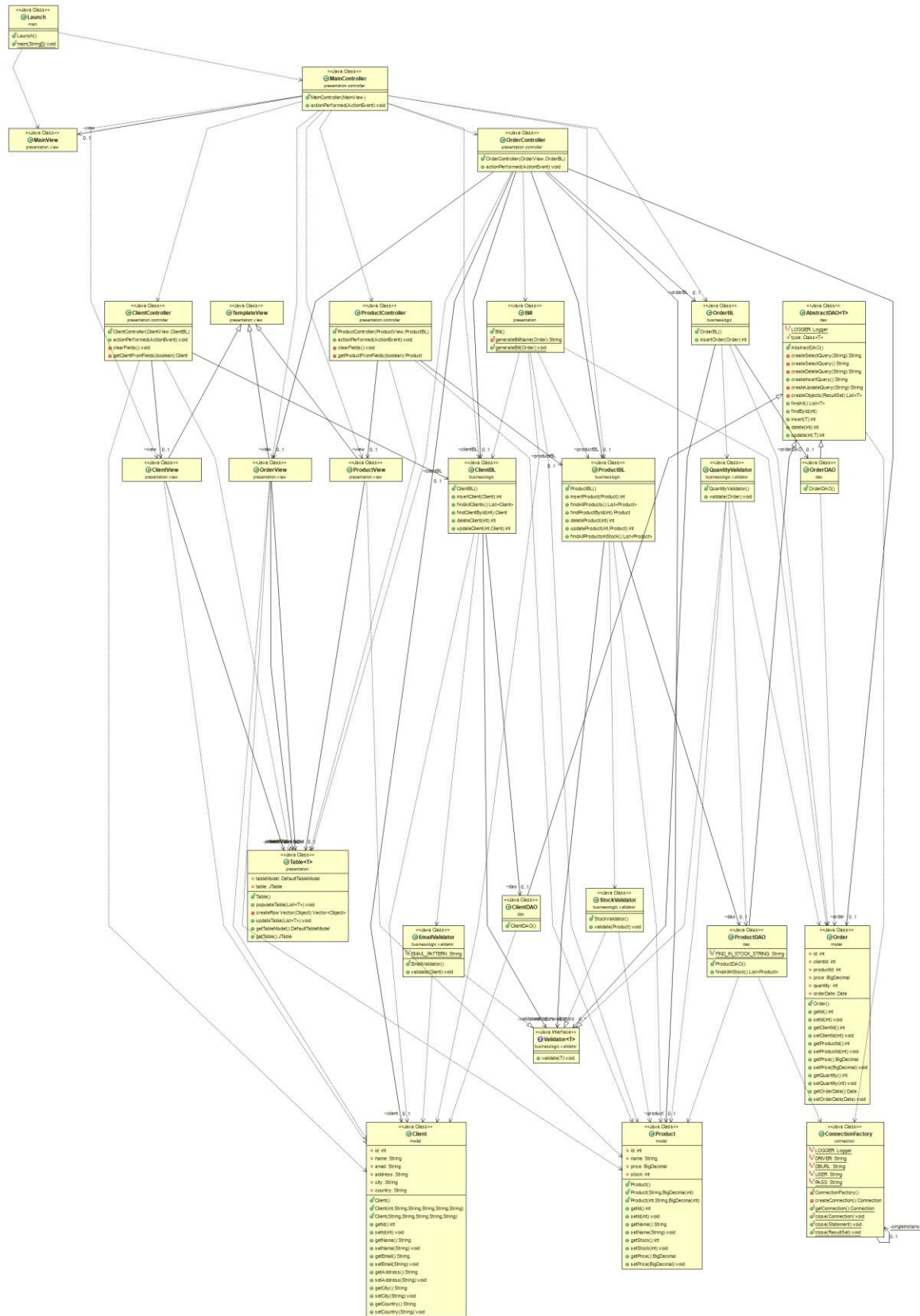
- diagrama de pachete și subpachete



- diagrama de clase simplificată – **fără** pachetul *presentation*



- diagrama de clase completă (png)



## 4. Implementarea temei

În acest capitol vor fi descrise toate clasele din aplicație împreună cu atributele și metodele cele mai importante.

### 4.1. Pachetul *model*

Conține modelele utilizate în aplicație, care vor fi mapate în baza de date relațională.

Clasa **Client** – reține datele unui client

#### *Attribute*

- id
- name – numele clientului
- email – adresa de email
- address – adresa (strada, număr)
- city – orașul
- country – țara

Clasa **Product** – reține datele unui produs

#### *Attribute*

- id
- name – numele produsului
- price – prețul unitar
- stock – stocul produsului (mai mare sau egal cu 0)

Clasa **Order** – reține datele unei comenzi

#### *Attribute*

- id
- clientId – ID-ul clientului
- productId – ID-ul produsului
- price – prețul total (prețul unitar al produsului  $\times$  cantitatea selectată)
- quantity – cantitatea
- orderDate – data în care s-a adăugat comanda

### 4.2. Pachetul *connection*

Clasa **ConnectionFactory**

#### *Metode*

- createConnection ( ) – creează o nouă conexiune cu baza de date
- getConnection ( ) – returnează o conexiune existentă
- close (Connection connection) – închide conexiunea *connection*
- close (Statement statement) – închide un statement *statement*
- close (ResultSet resultSet) – închide un set de rezultate *resultSet*

### 4.3. Pachetul *dao*

Conține clasele DAO pentru fiecare model mapat.

Clasa **AbstractDAO** – cea mai importantă clasă din pachet

#### *Attribute*

- type – va reține tipul de model (Client, Product, Order)

#### *Metode*

- createSelectQuery ( ) – construiește un String în funcție de *type*, care va conține instrucțiunea pentru afișarea tuturor înregistrărilor dintr-un tabel (*select \* from tabel*)

- `createSelectQuery (String field)` - construiește un String în funcție de *type*, care va conține instrucțiunea pentru afișarea tuturor înregistrărilor care îndeplinesc o condiție (*select \* from tabel where field = ?*)
- `createInsertQuery ()` – construiește un String în funcție de *type*, care va conține instrucțiunea pentru adăugarea unei noi înregistrări într-un tabel (*insert into tabel(...) values (...)*)
- `createUpdateQuery (String field)` - construiește un String în funcție de *type*, care va conține instrucțiunea pentru actualizarea unei înregistrări într-un tabel, unde se îndeplinește o condiție (*update tabel set...*)
- `createDeleteQuery (String field)` - construiește un String în funcție de *type*, care va conține instrucțiunea pentru ștergerea unei înregistrări, unde se îndeplinește o condiție (*delete from tabel where field = ?*)
- `createObjects (ResultSet resultSet)` – returnează, în caz de succes, o listă de obiecte de tip *type* mapate dintr-un ResultSet
  - lista inițial este goală
  - cât timp mai exista rezultate în ResultSet
    - se creează o noua instanță a tipului *type*
    - pentru fiecare atribut al tipului *type*, returnează obiectul corespunzător din resultSet și setează valoarea aceluiși atribut din instanță cu valoarea obiectului din resultSet (folosind `invoke`)
    - adaugă instanța în listă (s-a obținut un obiect de tip *type*)
- `findAll ()` – returnează lista tuturor obiectelor de tip *type* mapate din tabelul corespunzător
- `findById (int id)` – returnează obiectul de tip *type* din tabelul corespunzător care are ID-ul *id*
- `insert (T object)` – adaugă o înregistrare cu valorile din obiectul *object* în tabelul corespunzător și returnează ID-ul generat de baza de date la adăugarea obiectului *object* în caz de succes
- `delete (int id)` – șterge din tabelul corespunzător înregistrarea cu ID-ul *id* și îl returnează dacă operația s-a efectuat cu succes
- `update (int id, T newValues)` – actualizează, în tabelul corespunzător, valorile câmpurilor înregistrării cu ID-ul *id*, cu valorile atributelor din *newValues* și returnează ID-ul în caz de succes

Clasele **ClientDAO**, **ProductDAO** și **OrderDAO** extind clasa **AbstractDAO** cu tipul de model corespunzător. Astfel, moștenesc toate metodele menționate anterior pentru tipul lor de model.

#### Clasa **ProductDAO**

##### Metode

- `findAllInStock ()` – returnează o listă de produse care au stocul mai mare decât 0

#### 4.4. Pachetul *businessLogic*

Conțin validatorii și clasele de logică pentru fiecare tip de model. Fiecare metodă dintr-o clasă BL apelează metoda corespunzătoare din DAO (ex: `insertClient` apelează `insert` din **ClientDAO**, etc.)

Interfața **Validator** – tip generic T

##### Metode

- `validate (T object)` – *mod de funcționare*: va arunca o excepție dacă atributele obiectului *object* încalcă anumite reguli

Clasa **EmailValidator** – implementează Validator pe Client

##### Metode

- `validate (Client object)` – aruncă o excepție dacă emailul clientului nu respectă un pattern prestabilit

Clasa **StockValidator** – implementează Validator pe Produs

##### Metode

- `validate (Product object)` – aruncă o excepție dacă stocul produsului e mai mic sau egal cu 0

Clasa **QuantityValidator** – implementează Validator pe Order

##### Metode

- `validate (Client object)` – aruncă o excepție dacă cantitatea comandată este mai mică sau egală cu 0 sau depășește stocul produsului comandat

#### Clasa **ClientBL**

##### **Metode**

- `findAllClients ()` – returnează o listă cu toți clienții din baza de date
- `findClientById (int id)` – returnează clientul care are ID-ul `id`
- `insertClient (Client client)` – verifică dacă atributele lui client sunt conforme cu ajutorul validatorilor și în caz de succes îl adaugă în baza de date
- `deleteClient (int id)` – șterge clientul cu ID –ul `id` din baza de date și returnează ID-ul în caz de succes
- `updateClient (int id, Client newValues)` – actualizează înregistrarea cu ID-ul `id` din tabelul de clienți cu valorile din `newValues` și returnează ID-ul în caz de succes

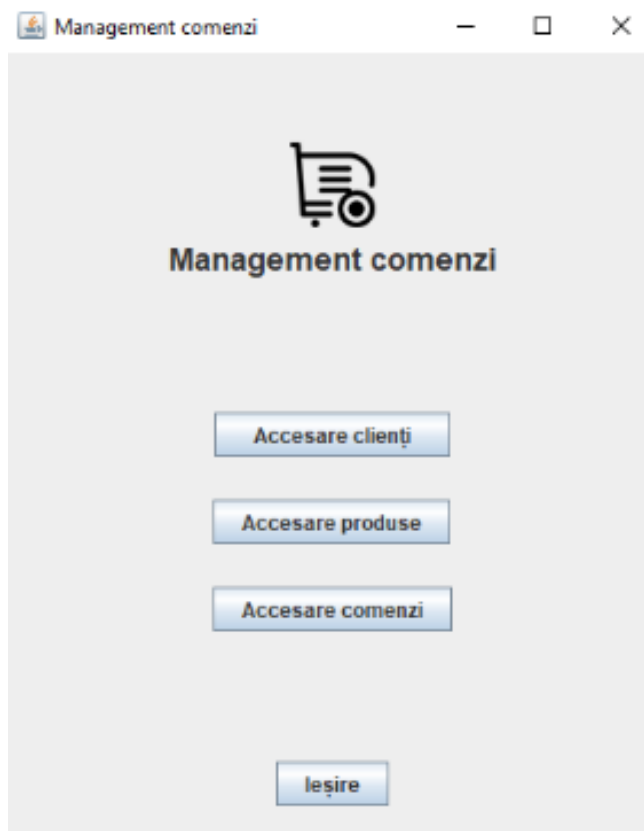
Metodele de *insert* și *update* vor propaga excepțiile aruncate de validatori în controller dacă nu sunt respectate regulile de business.

Clasele **ProductBL** și **OrderBL** sunt analoge, cu excepția numelor metodelor, care sunt modificate corespunzător.

#### **4.5. Pachetul *presentation***


Conține view-urile pentru fiecare secțiune din aplicație, controller-ele corespunzătoare și încă două clase pentru tabele și generarea de facturi.

##### Clasa **MainView**



##### Clasa **MainController**

### Clasa **ClientView** – extinde TemplateView



Listare

Adăugare

Actualizare

Ștergere

Închidere


Clienți
— □ ×

### Listă clienți

id	name	email	address	city	country
1	Doru Pop	dorupop@email.c...	str. Transilvaniei, n...	Bucuresti	Romania
2	Adina Stan	adinastan@email...	str. Republicii, nr 40	Cluj-Napoca	Romania
3	John Smith	jsmith@email.com	Park Street, nr. 16	London	Regatul Unit
4	Hans Weiss	hweiss@email.com	Eulengasse, nr. 3	Frankfurt am Main	Germania

### Clasa **ClientController**

### Clasa **ProductView** – extinde TemplateView



Listare

Adăugare

Actualizare

Ștergere

Închidere

Produce
— □ ×


### Listă produse

id	name	price	stock
1	Playstation 5	4999.99	49
2	Gaming Keyboard	329.99	170
3	Smartphone Samsung Galax...	2260.00	300
4	LED TV Starlight	599.99	0
5	Gaming Laptop	7299.99	5

### Clasa **ProductController**

### Clasa **OrderView** – extinde TemplateView


Comenzi



### Selectare client

id	name	email	address	city	country
1	Doru Pop	dorupop@email.c...	str. Transilvaniei, n.	Bucuresti	Romania
2	Adina Stan	adinastan@email...	str. Republicii, nr.40	Cluj-Napoca	Romania
3	John Smith	jsmith@email.com	Park Street, nr. 16	London	Regatul Unit
4	Hans Weiss	hweiss@email.com	Eulengasse, nr.3	Frankfurt am Main	Germania
25	Ion Pop	popion@email.com	str. Ostrovului, bl.3...	Satu Mare	Romania

Comenzi



### Finalizare comandă

Client: Doru Pop  
Email: dorupop@email.com  
Adresă: str. Transilvaniei, nr.2, București, Romania

Produs:  
Cod: 1  
Denumire: Playstation 5  
Preț/bucată: 4999.99  
Stoc: 50 bucăți

Număr bucăți:

### Clasa **OrderController**

Clasa **TemplateView** – extinde JFrame; este o clasă șablon pentru ClientView, ProductView și OrderView. Pentru eficiență și *user experience*, toate aceste clase menționate vor avea un aspect similar.

Clasa **Table** – clasă cu tip generic – are ca atribute un DefaultTableModel și un JTable cu modelul respectiv.

**Metode**

- populateTable (List<T> list) - generează coloanele tabelului, care sunt atributele unui obiect de tip *T* și populează tabelul cu obiectele din lista *list*
- createRowVector(Object object) - returnează un vector de obiecte, ale cărui elemente sunt valorile atributelor obiectului *object*
- updateTable (List<T> list) – actualizează tabelul; analog cu populateTable, cu excepția faptului că nu se mai generează încă o dată capul de tabel

Clasa **Bill** – conține metoda care generează factura unei comenzi

**Metode**

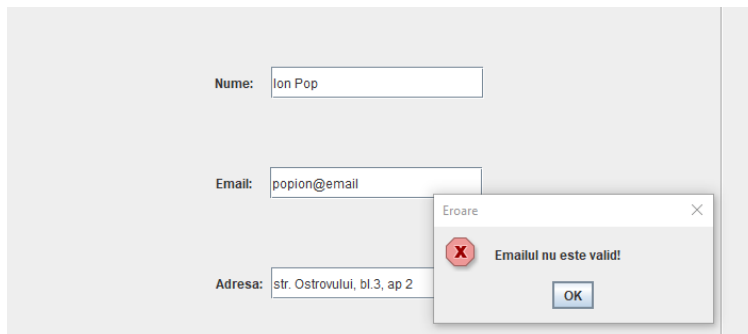
- generateBillName (Order order) – generează numele fișierului care va fi creat în funcție de ID-ul lui *order*
- generateBill (Order order) generează factura unei comenzi *order*; Factura va conține:
  - numele, emailul și adresa completă a clientului
  - ID-ul, denumirea și prețul unitar al produsului
  - cantitatea comandată
  - totalul comenzii (câmpul *price* din Order)

#### 4.6. Pachetul *main*

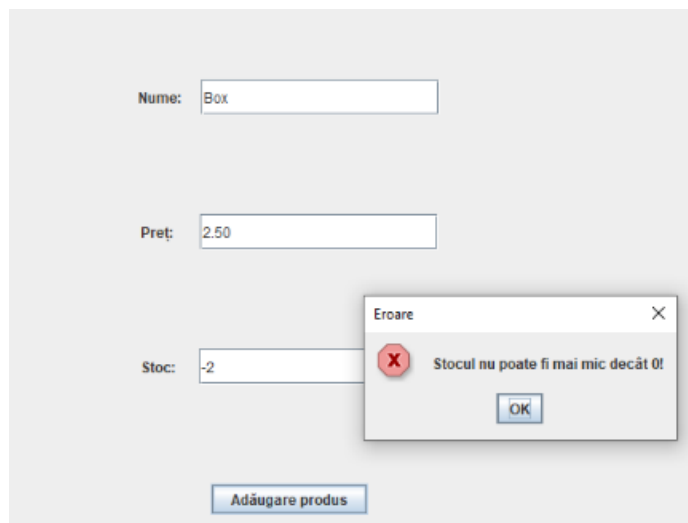
Clasa **Launch** – se execută aplicația

## 5. Testarea temei și rezultate

Testări în aplicație ale excepțiilor:



The screenshot shows a registration form with three input fields: 'Nume:' containing 'Ion Pop', 'Email:' containing 'popion@email', and 'Adresa:' containing 'str. Ostrovului, bl.3, ap 2'. An error dialog box titled 'Eroare' is displayed over the email field, showing a red 'X' icon and the message 'Emailul nu este valid!' with an 'OK' button.



The screenshot shows a product form with three input fields: 'Nume:' containing 'Box', 'Preț:' containing '2.50', and 'Stoc:' containing '-2'. An error dialog box titled 'Eroare' is displayed over the stock field, showing a red 'X' icon and the message 'Stocul nu poate fi mai mic decât 0!' with an 'OK' button. At the bottom of the form is a button labeled 'Adăugare produs'.



Comenzi



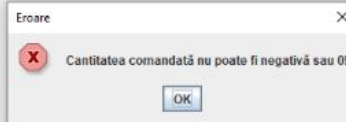
Închidere

### Finalizare comandă

Client: Doru Pop  
Email: dorupop@email.com  
Adresă: str. Transilvaniei, nr.2, Bucuresti, Romania

Produs:  
Cod:1  
Denumire: Playstation 5  
Preț/bucată: 4999.99  
Stoc: 50 bucăți

Număr bucăți: -3



< Înapoi Finalizare

Comenzi



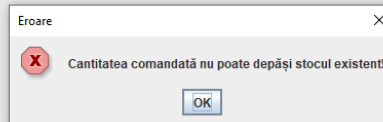
Închidere

### Finalizare comandă

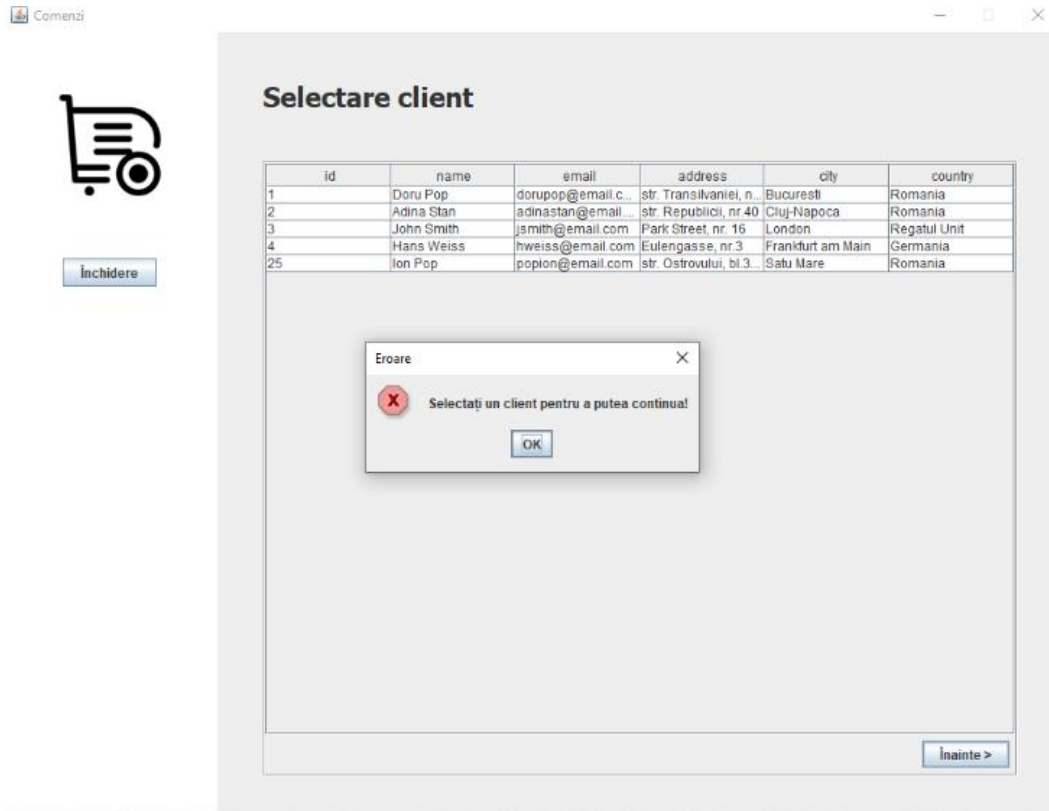
Client: Hans Weiss  
Email: hweiss@email.com  
Adresă: Eulengasse, nr.3, Frankfurt am Main, Germania

Produs:  
Cod: 3  
Denumire: Smartphone Samsung Galaxy A52  
Preț/bucată: 2260.00  
Stoc: 300 bucăți

Număr bucăți: 400



< Înapoi Finalizare



## 6. Concluzii și dezvoltări ulterioare

Realizând această temă, am învățat:

- un nou model arhitectural
- cum să utilizez *Java Reflection*
- cum să utilizez tipurile generice
- cum să folosesc `PrintWriter` cu diferite formate pentru fișiere
- cum să folosesc `CardLayout` pentru interfață

În versiuni ulterioare ale aplicației, aș dori să mai adaug validatoare pentru Business Logic Layer și să implementez lucruri noi pentru secțiunea de comenzi în aplicație.

## 7. Webografie

1. De pe Microsoft Teams, echipa TP Seria 1 An 2 - 2021  
ASSIGNMENT\_3\_SUPPORT\_PRESENTATION.pdf
2. JavaTPoint, "Java PrintWriterClass" - <https://www.javatpoint.com/java-printwriter-class>;
3. JavaTPoint, "Java CardLayout" - <https://www.javatpoint.com/CardLayout>;
4. HowToDoInJava, "Java email validation using regex" - <https://howtodoinjava.com/java/regex/java-regex-validate-email-address/>;
5. GeeksforGeeks, "BigDecimal Class" - <https://www.geeksforgeeks.org/bigdecimal-class-java/>;
6. Baeldung, "JAVADOC" - <https://www.baeldung.com/javadoc>;
7. MySQL, "SQL Data Export and Import Wizard" - <https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>;
8. w3schools, "Java Date and Time" - [https://www.w3schools.com/java/java\\_date.asp](https://www.w3schools.com/java/java_date.asp);