MINISTERUL EDUCAȚIEI ȘI CERCETĂRII
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

TEHNICI DE PROGRAMRE FUNDAMENTALE

TEMA 3

Managementul comenzilor

Dragoș-Bogdan Lazea, grupa 30223

An universitar: 2020 – 2021

1. Obiectivul temei

Obiectivul principal al acestei teme îl reprezintă proiectarea și implementarea unei aplicații care să simuleze funcționarea aplicațiilor construite pe fundamentul unei baze de date, fiind dedicate creării și înregistrării unor comenzi de către mai mulți clienți și dispunând de o interfață grafică intuitivă pentru a facilita interacțiunea cu orice tip de utilizator.

Obiectivele secundare ale aceste teme sunt prezentate în tabelul de mai jos:

Obiectiv secundar	Descriere	Secțiune
1. Analiza problemei	- presupune analiza și delimitarea tuturor cerințelor pe care aplicația trebuie să le îndeplinească;	2. Analiza problemei, modelare, scenarii, cazuri de utilizare
2. Identificarea scenariilor	- presupune identificarea și analiza tuturor scenariilor în care s-ar putea regăsi aplicația în funcție de valorile de intrare introduse de către utilizator (cazuri limită, excepții, funcționare corespunzătoare);	2. Analiza problemei, modelare, scenarii, cazuri de utilizare
3. Identificarea cazurilor de utilizare	- presupune identificarea situațiilor în care aplicația își justifică funcționalitatea;	2. Analiza problemei, modelare, scenarii, cazuri de utilizare
4. Proiectarea propriu-zisă	- presupune parcurgerea etapelor de sintetizare și proiectare a aplicației conform paradigmei programării orientat pe obiecte, prin realizarea de diagrame UML, proiectarea claselor, imaginarea interfeței grafice cu utilizatorul etc.	3. Proiectare
5. Implementarea aplicației	- presupune dezvoltarea propriu zisă a claselor și metodelor aferente acestora în cod scris în limbajul de programare Java;	4. Implementare
6. Testarea aplicației implementate	- presupune testarea funcționalității aplicației în toate situațiile posibile de funcționare sintetizate în cadrul cazurilor de utilizare;	5. Rezultate

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

a. Analiza problemei

Problema de soluționat este reprezentată de faptul că în lumea reală, sistemele bazate pe politici de marketing dovedesc adesea ineficiență în ceea ce privește înregistrarea comenzilor sau păstrarea consistentă a datelor referitoare la clienți și produse.

Prin urmare, a fost identificat, în urma analize problemei menționate, următorul cadru de cerințe funcționale:

- Aplicația trebuie să poată permite introducerea numărului de noi clienți, caracterizați prin nume, adresă, adresă de e-mail și vârstă, și de produse disponibile, identificate prin nume, preț și cantitate disponibilă în stoc;
- Aplicația trebuie să informeze utilizatorul dacă inputul introdus este invalid, i. e. e-mailul nu respectă șablonul universal pentru o adresă de e-mail, clienții înregistrați nu sunt majori sau cantitatea disponibilă pentru un produs nu este un număr natural;
- Aplicația trebuie să poată permite selectarea actualizarea și ștergerea informațiilor referitoare la clienți și produse și menținerea consistentă a acestora;
- Aplicația trebuie să permită selectarea unui client existent în baza de date, selectarea unui tip de produs din stoc, introducerea unei cantități dorite de către client și plasarea comenzii;
- Aplicația trebuie să permită vizualizarea tuturor clienților înregistrați și a tuturor produselor disponibile spre a fi achiziționate;
- Aplicația trebuie să informeze utilizatorul dacă în stoc nu se află suficiente produse pentru a putea satisface cererea acestuia.

Cadrul cerințelor nefuncționale conturate în jurul problemei de rezolvat este:

- Aplicația trebuie să dispună de o interfață grafică intuitivă.
- Aplicația trebuie să fie ușor de utilizat pentru orice tip de utilizator, indiferent de domeniul de activitate al acestuia.

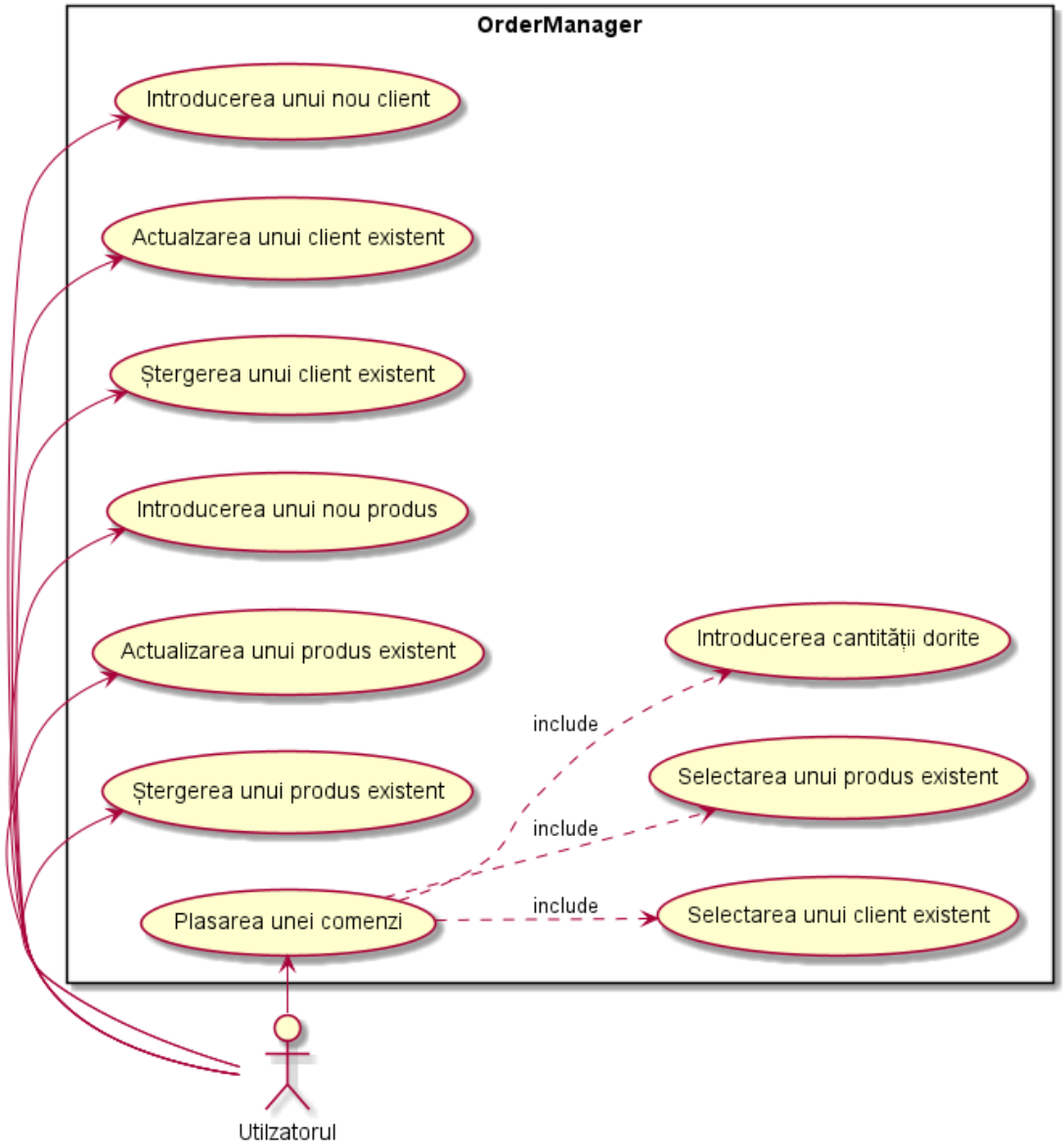
b. Modelare

Aplicația a fost proiectată pe modelul unui sistem cu trei intrări compuse și o ieșire, intrările fiind reprezentate client – având atributele nume, adresă, e-mail, vârstă – produs – caracterizat prin nume, preț și cantitate disponibilă în stoc – și cantitatea dorită spre a fi cumpărată de către client, în timp ce ieșirea este reprezentată de factura generată în urma plasării și înregistrării comenzii în baza de date.

Modelul problemei este astfel translatat într-un model orientat pe obiecte, ce are la bază clasele Client, Product și Order prin intermediul cărora se vor defini și transpune în modelul obiectual intrările și ieșirile aplicației, definite pe baza modelului din lumea reală.

c. Cazuri de utilizare și scenarii posibile

Diagrama cazurilor de utilizare este prezentată mai jos:



În cele ce urmează vor fi descrise cazurile de utilizare alte aplicației dezvoltate:

Use Case: Introducerea unui nou client
Actorul principal: Utilizatorul
Scenariul de succes:

1. Utilizatorul introduce corect informațiile necesare înregistrării unui nou client.
2. Utilizatorul înregistrează un nou client apăsând butonul „Register client”.
3. Aplicația permite vizualizarea clientului inserat în tabelul ce conține toți clienții existenți din fereastra Clients.

Scenarii alternative (posibile erori): Cel puțin una dintre câmpurile Email și Age nu respectă formatul impus: email trebuie să respecte șablonul universal pentru o adresă de email validă, iar age trebuie să fie un număr natural mai mare sau egal cu 18.

1. Utilizatorul este notificat prin apariția unei ferestre cu un mesaj de eroare corespunzător excepției generate.
2. Scenariul se întoarce la pasul de introducere a informațiilor necesare simulării de către utilizator.

Use Case: Actualizarea unui client existent
Actorul principal: Utilizatorul
Scenariul de succes:

1. Utilizatorul selectează un client existent din tabelul ce conține totalitatea clienților înregistrați.
2. Utilizatorul modifică conținutul unor câmpuri din cadrul clientului selectat, cu excepția câmpului id care nu poate fi modificat.
3. Utilizatorul apasă butonul „Edit client”.
4. Aplicația permite vizualizarea modificărilor efectuate asupra clientului în tabelul ce conține toți clienții existenți din fereastra Clients.

Scenarii alternative (posibile erori): Cel puțin una dintre câmpurile corespunzătoare atributelor Email și Age nu respectă formatul impus: email trebuie să respecte șablonul universal pentru o adresă de email validă, iar age trebuie să fie un număr natural mai mare sau egal cu 18.

1. Utilizatorul este notificat prin apariția unei ferestre cu un mesaj de eroare corespunzător excepției generate.
2. Nu se efectuează modificări asupra clientului selectat.

Use Case: Ștergerea unui client existent
Actorul principal: Utilizatorul
Scenariul de succes:

1. Utilizatorul selectează un client din tabelul ce conține toți clienții.
2. Utilizatorul apasă butonul „Delete client”.
3. Aplicația permite vizualizarea modificărilor efectuate în tabelul ce conține toți clienții existenți din fereastra Clients.

Use Case: Introducerea unui nou produs
Actorul principal: Utilizatorul
Scenariul de succes:

1. Utilizatorul introduce corect informațiile necesare înregistrării unui nou produs.
2. Utilizatorul înregistrează un nou client apăsând butonul „Register product”.
3. Aplicația permite vizualizarea produsului inserat în tabelul ce conține toate produsele existente din fereastra Products.

Scenarii alternative (posibile erori): Câmpul pentru cantitatea disponibilă în stoc este mai mic decât 0.

1. Utilizatorul este notificat prin apariția unei ferestre cu un mesaj de eroare corespunzător excepției generate.
2. Scenariul se întoarce la pasul de introducere a informațiilor necesare simulării de către utilizator.

Use Case: Actualizarea unui produs existent
Actorul principal: Utilizatorul
Scenariul de succes:

1. Utilizatorul selectează un produs existent din tabelul ce conține totalitatea produselor înregistrate.
2. Utilizatorul modifică conținutul unor câmpuri din cadrul produsului selectat, cu excepția câmpului id care nu poate fi modificat.
3. Utilizatorul apasă butonul „Edit product”.
4. Aplicația permite vizualizarea modificărilor efectuate asupra produsului în tabelul ce conține toate produsele existente din fereastra Products.

Scenarii alternative (posibile erori): Câmpul corespunzător cantității disponibile în stoc este un număr mai mic decât 0.

1. Utilizatorul este notificat prin apariția unei ferestre cu un mesaj de eroare corespunzător excepției generate.
2. Nu se efectuează modificări asupra produsului selectat.

Use Case: Plasarea unei comenzi
Actorul principal: Utilizatorul
Scenariul de succes:

1. Utilizatorul selectează un client din tabelul ce conține toți clienții, aflat în fereastra Orders. (sub-use case I)
2. Utilizatorul selectează un produs din tabelul ce conține toate produsele, aflat în fereastra Orders. (sub-use case II)
3. Utilizatorul introduce o cantiate dorită mai mică sau egală cu cantitatea disponibilă pentru produsul respectiv. (sub-use case III)
4. Utilizatorul apasă butonul „Place order”.
5. Comanda este plasată în baza de date, iar fișierul .txt ce conține factura este generat.

Scenarii alternative (posibile erori): Câmpul corespunzător cantității disponibile în stoc este un număr mai mic decât cantitatea dorită de către client.

1. Utilizatorul este notificat prin apariția unei ferestre cu un mesaj de eroare corespunzător excepției generate.
2. Nu se înregistrează comanda, iar factura nu este generată.
3. Scenariu se întoarce la pasul introducerii datelor necesare comenzii.

3. Proiectare

Primul nivel de proiectare îl reprezintă imaginea de ansamblu asupra aplicației prin realizarea schemei bloc a aplicației (cutia neagră a sistemului), care poate fi vizualizată mai jos.

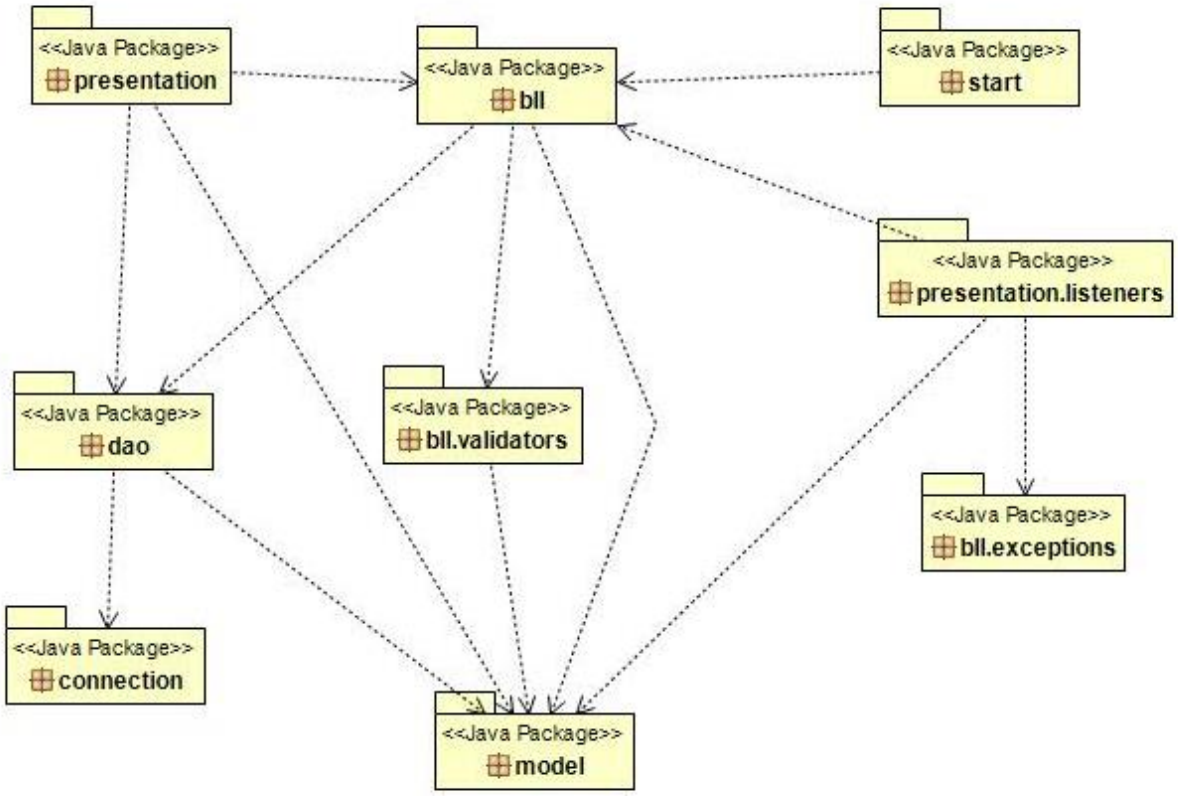
Următoarea etapă a proiectării este reprezentată de divizarea aplicației în subaplicații cu funcționalități specializate. Subsistemele componente ale aplicației fiind organizate conform unui pattern arhitectural de tip **Layered Architecture**, compus din: **dataAccessLayer**, **businessLayer**, **model** și **presentation,** următorul nivel de divizare al aplicației putând fi identificat în diagrama de pachete prezentată mai jos.

Modelul încapsulează datele specifice aplicației și definește obiectele, logica și computațiile care stochează, manipulează și procesează datele furnizate de utilizator, în timp ce vederea redă conținutul modelului, fiind responsabilă de realizarea interfeței grafice cu utilizatorul. Subaplicația definită în dataAccessLayer face posibilă accesarea și actualizarea informației stocate în baza de date asociată aplicației, iar businessLayer definește operațiile ce se pot efectua asupra bazei de date folosind modelul

obiectual descris în cadrul aplicației, conținând la rândul său un subpachet responsabil de validarea datelor și unul pentru semnalarea excepțiilor generate.

De asemenea, s-a dovedit necesară realizarea pachetului **connection**, responsabil de realizarea conexiunii cu baza de date asociată.

Astfel, aplicația va fi compus din cele patru componente menționate mai sus, formate la rândul lor din subcomponente, interconectate astfel încât să formeze un ansamblu unitar, fapt ce poate fi observat în următoarea diagramă de pachete.



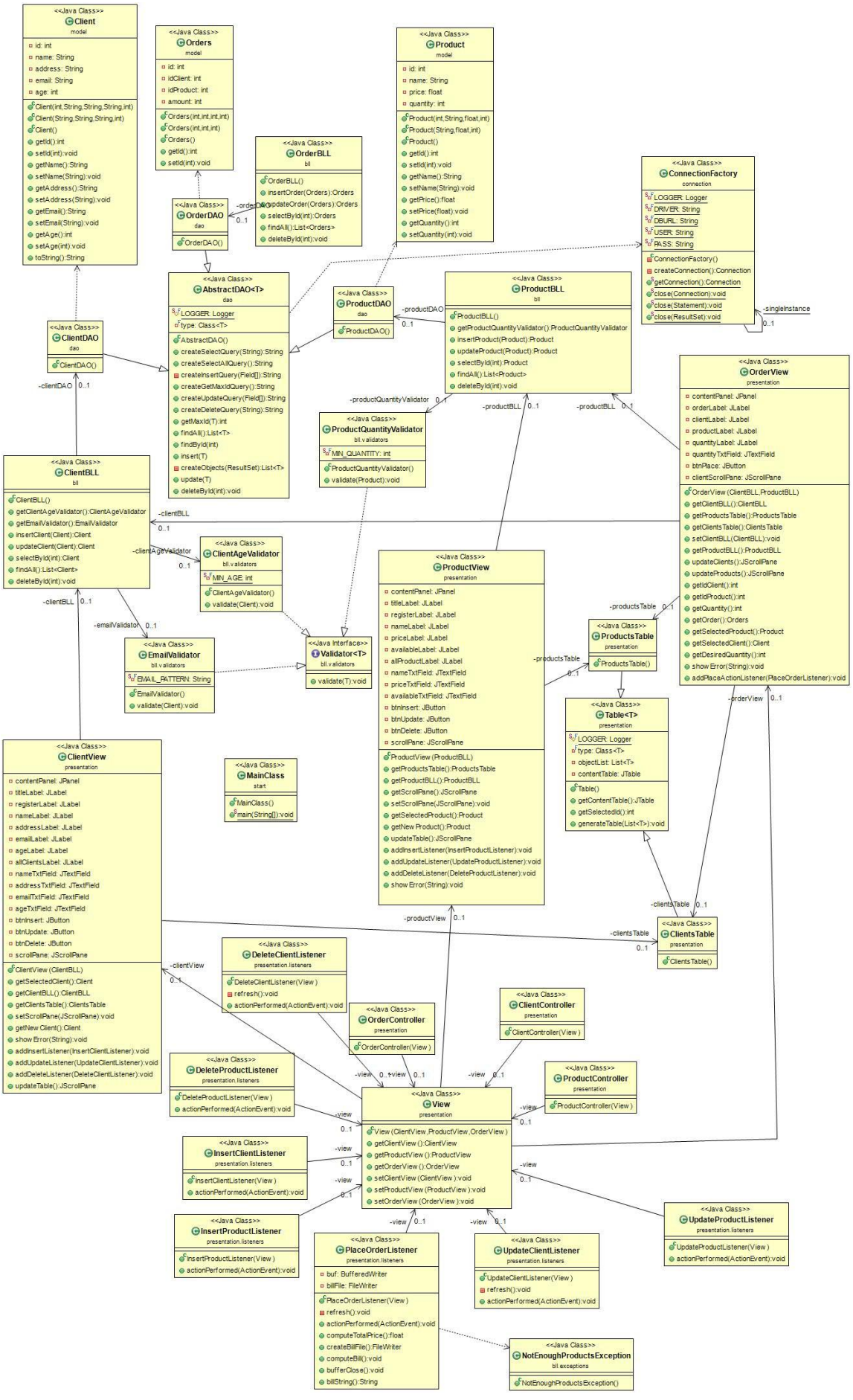
Trecând la cea de-a treia etapă de proiectare, se continuă descompunerea problemei și astfel se definesc cele mai de jos unități din descompunerea aplicației conform paradigmei programării orientate pe obiecte (POO), și anume clasele definite de dezvoltator.

Astfel, după cum a fost menționat anterior, fiecare pachet identificat în descompunerea aplicației are o specializare distinctă, aceasta fiind implementată propriu zis prin intermediul claselor componente ale pachetului.

Descrierea tuturor claselor va fi detaliată în secțiunea dedicată implementării aplicației, împreună cu metodele importante și definitorii fiecărei clase din componența proiectului.

Dintre algoritmii utilizați la proiectarea soluției, relevanți pentru analiză în contextul dezvoltării aplicației se dovedesc algoritmii pentru construirea interogărilor în limbajul MySQL. Acestea sunt construite în primă fază sub forma unor șiruri de caractere, fiind apoi transpuse în statement-uri care vor fi executate cu ajutorul metodelor de tip execute și vor genera rezultate de tip ResultSet.

În cele ce urmează va fi expusă diagrama UML de clase ale aplicației, denumirile acestora fiind sugestive și inspirate din domeniul problemei.


<<Java Class>> Client
model
id: int
name: String
address: String
email: String
age: int
Client(int,String,String,String,int)
Client(String,String,String,int)
Client()
getId():int
setId(int):void
getName():String
setName(String):void
getAddress():String
setAddress(String):void
getEmail():String
setEmail(String):void
getAge():int
setAge(int):void
toString():String
<<Java Class>> Orders
model
id: int
idClient: int
idProduct: int
amount: int
Orders(int,int,int,int)
Orders(int,int,int)
Orders()
getId():int
setId(int):void
<<Java Class>> Product
model
id: int
name: String
price: float
quantity: int
Product(int,String,float,int)
Product(String,float,int)
Product()
getId():int
setId(int):void
getName():String
setName(String):void
getPrice():float
setPrice(float):void
getQuantity():int
setQuantity(int):void
<<Java Class>> OrderBLL
bll
OrderBLL()
insertOrder(Orders):Orders
updateOrder(Orders):Orders
selectById(int):Orders
findAll():List<Orders>
deleteById(int):void
<<Java Class>> OrderDAO
dao
OrderDAO()
<<Java Class>> ConnectionFactory
connection
LOGGER: Logger
DRIVER: String
DBURL: String
USER: String
PASS: String
ConnectionFactory()
createConnection():Connection
getConnection():Connection
close(Connection):void
close(Statement):void
close(ResultSet):void
<<Java Class>> AbstractDAO<T>
dao
LOGGER: Logger
type: Class<T>
AbstractDAO()
createSelectQuery(String):String
createSelectAllQuery():String
createInsertQuery(Field[]):String
createGetMaxIdQuery():String
createUpdateQuery(Field[]):String
createDeleteQuery(String):String
getMaxId(T):int
findAll():List<T>
findById(int)
insert(T)
createObjects(ResultSet):List<T>
update(T)
deleteById(int):void
<<Java Class>> ProductDAO
dao
ProductDAO()
<<Java Class>> ProductBLL
bll
ProductBLL()
getProductQuantityValidator():ProductQuantityValidator
insertProduct(Product):Product
updateProduct(Product):Product
selectById(int):Product
findAll():List<Product>
deleteById(int):void
<<Java Class>> ClientDAO
dao
ClientDAO()
<<Java Class>> ProductQuantityValidator
bll.validators
MIN_QUANTITY: int
ProductQuantityValidator()
validate(Product):void
<<Java Class>> OrderView
presentation
contentPanel: JPanel
orderLabel: JLabel
clientLabel: JLabel
productLabel: JLabel
quantityLabel: JLabel
quantityTxtField: JTextField
btnPlace: JButton
clientScrollPane: JScrollPane
OrderView (ClientBLL,ProductBLL)
getClientBLL():ClientBLL
getProductsTable():ProductsTable
getClientsTable():ClientsTable
setClientBLL(ClientBLL):void
getProductBLL():ProductBLL
updateClients():JScrollPane
updateProducts():JScrollPane
getIdClient():int
getIdProduct():int
getQuantity():int
getOrder():Orders
getSelectedProduct():Product
getSelectedClient():Client
getDesiredQuantity():int
show Error(String):void
addPlaceActionListener(PlaceOrderListener):void
<<Java Class>> ClientBLL
bll
ClientBLL()
getClientAgeValidator():ClientAgeValidator
getEmailValidator():EmailValidator
insertClient(Client):Client
updateClient(Client):Client
selectById(int):Client
findAll():List<Client>
deleteById(int):void
<<Java Class>> ClientAgeValidator
bll.validators
MIN_AGE: int
ClientAgeValidator()
validate(Client):void
<<Java Class>> ProductView
presentation
contentPanel: JPanel
titleLabel: JLabel
registerLabel: JLabel
nameLabel: JLabel
priceLabel: JLabel
availableLabel: JLabel
allProductLabel: JLabel
nameTxtField: JTextField
priceTxtField: JTextField
availableTxtField: JTextField
btnInsert: JButton
btnUpdate: JButton
btnDelete: JButton
scrollPane: JScrollPane
ProductView (ProductBLL)
getProductsTable():ProductsTable
getProductBLL():ProductBLL
getScrollPane():JScrollPane
setScrollPane(JScrollPane):void
getSelectedProduct():Product
getNew Product():Product
updateTable():JScrollPane
addInsertListener(InsertProductListener):void
addUpdateListener(UpdateProductListener):void
addDeleteListener(DeleteProductListener):void
show Error(String):void
<<Java Class>> ProductsTable
presentation
ProductsTable()
<<Java Class>> EmailValidator
bll.validators
EMAIL_PATTERN: String
EmailValidator()
validate(Client):void
<<Java Interface>> Validator<T>
bll.validators
validate(T):void
<<Java Class>> Table<T>
presentation
LOGGER: Logger
type: Class<T>
objectList: List<T>
contentTable: JTable
Table()
getContentTable():JTable
getSelectedId():int
generateTable(List<T>):void
<<Java Class>> ClientView
presentation
contentPanel: JPanel
titleLabel: JLabel
registerLabel: JLabel
nameLabel: JLabel
addressLabel: JLabel
emailLabel: JLabel
ageLabel: JLabel
allClientsLabel: JLabel
nameTxtField: JTextField
addressTxtField: JTextField
emailTxtField: JTextField
ageTxtField: JTextField
btnInsert: JButton
btnUpdate: JButton
btnDelete: JButton
scrollPane: JScrollPane
ClientView (ClientBLL)
getSelectedClient():Client
getClientBLL():ClientBLL
getClientsTable():ClientsTable
setScrollPane(JScrollPane):void
getNew Client():Client
show Error(String):void
addInsertListener(InsertClientListener):void
addUpdateListener(UpdateClientListener):void
addDeleteListener(DeleteClientListener):void
updateTable():JScrollPane
<<Java Class>> MainClass
start
MainClass()
main(String[]):void
<<Java Class>> ClientsTable
presentation
ClientsTable()
<<Java Class>> DeleteClientListener
presentation.listeners
DeleteClientListener(View)
refresh():void
actionPerformed(ActionEvent):void
<<Java Class>> OrderController
presentation
OrderController(View)
<<Java Class>> ClientController
presentation
ClientController(View)
<<Java Class>> DeleteProductListener
presentation.listeners
DeleteProductListener(View)
actionPerformed(ActionEvent):void
<<Java Class>> ProductController
presentation
ProductController(View)
<<Java Class>> View
presentation
View (ClientView,ProductView,OrderView)
getClientView ():ClientView
getProductView ():ProductView
getOrderView ():OrderView
setClientView (ClientView):void
setProductView (ProductView):void
setOrderView (OrderView):void
<<Java Class>> InsertClientListener
presentation.listeners
InsertClientListener(View)
actionPerformed(ActionEvent):void
<<Java Class>> InsertProductListener
presentation.listeners
InsertProductListener(View)
actionPerformed(ActionEvent):void
<<Java Class>> PlaceOrderListener
presentation.listeners
buf: BufferedWriter
billFile: FileWriter
PlaceOrderListener(View)
refresh():void
actionPerformed(ActionEvent):void
computeTotalPrice():float
createBillFile():FileWriter
computeBill():void
bufferClose():void
billString():String
<<Java Class>> UpdateClientListener
presentation.listeners
UpdateClientListener(View)
refresh():void
actionPerformed(ActionEvent):void
<<Java Class>> UpdateProductListener
presentation.listeners
UpdateProductListener(View)
actionPerformed(ActionEvent):void
<<Java Class>> NotEnoughProductsException
bll.exceptions
NotEnoughProductsException()

4. Implementare

4.1. Clase, interfețe și metode definite

Denumirile claselor, ale variabilelor instanță și ale metodelor ce urmează a fi sintetizate și prezentate în cele ce urmează sunt sugestive, nefiind necesară explicitarea semnificației acestora.

În cele ce urmează se va explica utilitatea fiecărei clase implementate:

Package model

Class Summary

Class	Description
Client	Clasa pentru obiectul de tip Client
Orders	Clasa pentru obiectul de tip Order - a fost denumita Orders pentru a evita folosirea cuvantului rezervat order din limbajul MySQL
Product	Clasa pentru obiectul de tip Product

Package connection

Class Summary

Class	Description
ConnectionFactory	Clasa pentru realizarea conexiunii la baza de date

Package dao

Class Summary

Class	Description
AbstractDAO<T>	Clasa pentru operatiile pe baza de date - s-au folosit tipuri generice pentru a facilita utilizarea acestor operatii pentru toate cele trei tipuri de obiecte: Client, Product, Order
ClientDAO	Clasa concreta cu ajutorul careia se vor efectua operatiile din baza de date in tabela corespunzatoare obiectului de tip Client
OrderDAO	Clasa concreta cu ajutorul careia se vor efectua operatiile din baza de date in tabela corespunzatoare obiectului de tip Order
ProductDAO	Clasa concreta cu ajutorul careia se vor efectua operatiile din baza de date in tabela corespunzatoare obiectului de tip Product

Package bll

Class Summary

Class	Description
ClientBLL	Clasa in care s-au implementat operatiile pe obiectul de tip Client
OrderBLL	Clasa in care s-au implementat operatiile pe obiectul de tip Order
ProductBLL	Clasa in care s-au implementat operatiile pe obiectul de tip Product

Package bll.validators

Interface Summary

Interface	Description
Validator<T>	Interfata pentru validarea datelor introduse de utilizator

Class Summary

Class	Description
ClientAgeValidator	Clasa pentru validarea varstei clientilor
EmailValidator	Clasa pentru validarea email-ului clientilor
ProductQuantityValidator	Clasa pentru validarea cantitatii disponibile in stoc

Package bll.exceptions

Exception Summary

Exception	Description
NotEnoughProductsException	Clasa pentru definirea unei exceptii in cazul in care un client doreste sa cumpere o cantitate de produs mai mare decat cea disponibila in stoc

Package presentation

Class Summary

Class	Description
ClientController	Clasa de tip Controller atasata clasei ClientView
ClientsTable	Clasa concreta pentru tabelul cu produse din interfata grafica
ClientView	Clasa pentru fereastra din interfata grafica dedicata clientului
OrderController	Clasa de tip Controller atasata clasei ProductView
OrderView	Clasa pentru fereastra din interfata grafica dedicata comenzilor
ProductController	Clasa de tip Controller atasata clasei ProductView
ProductsTable	Clasa concreta pentru tabelul cu produse din interfata grafica
ProductView	Clasa pentru fereastra din interfata grafica dedicata produsului
Table<T>	Clasa pentru generarea tabelelor afisate in interfata grafica cu utilizatorul
View	Clasa pentru interfata grafica a aplicatiei cu cele trei ferestre atasate

Package presentation.listeners

Class Summary

Class	Description
DeleteClientListener	Clasa pentru definirea listenerului pentru butonul de stergere a unui client
DeleteProductListener	Clasa pentru definirea listenerului pentru butonul de stergere a unui produs
InsertClientListener	Clasa pentru definirea listenerului pentru butonul de inserare a unui client
InsertProductListener	Clasa pentru definirea listenerului pentru butonul de inserare a unui produs
PlaceOrderListener	Clasa pentru definirea listenerului pentru butonul de plasare a unei comenzi
UpdateClientListener	Clasa pentru definirea listenerului pentru butonul de acrualizare a unui client
UpdateProductListener	Clasa pentru definirea listenerului pentru butonul de acrualizare a unui produs

Package start

Class Summary

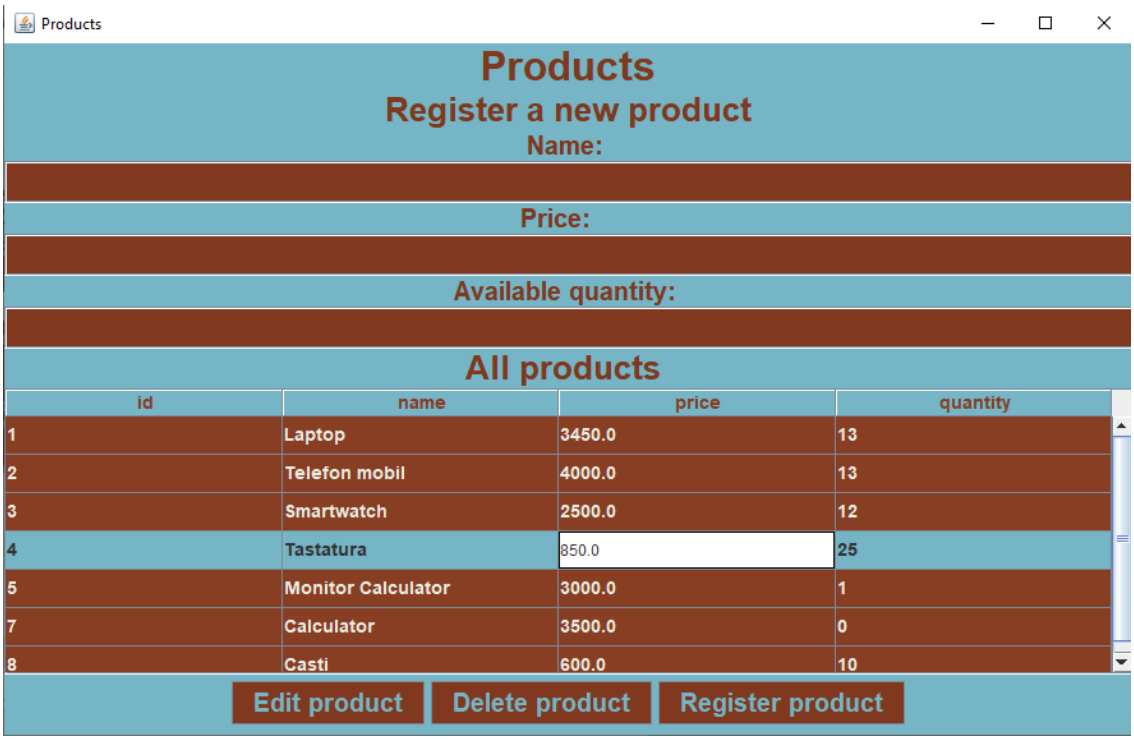
Class	Description
MainClass	Clasa principala ce contine o metoda main in care se vor instantia obiectele necesare utilizarii aplicatiei

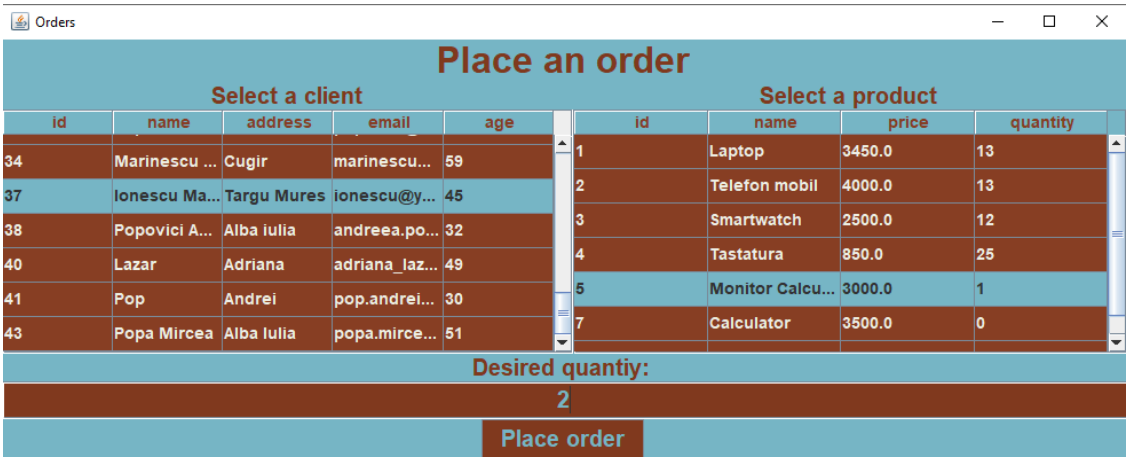
Descrierea implementării pentru toate metodele aferente fiecărei clase poate fi regăsită în cadrul JavaDoc, aceasta putând fi regăsită în directorul javadoc din folderul proiectului.

4.2. Interfața grafică cu utilizatorul (GUI)

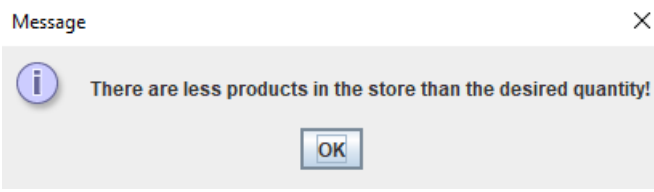
Aplicația dispune de o interfață grafică intuitivă, dezvoltată folosind Swing și fiind construită pe baza a trei JFrame: o fereastră dedicată introducerii și vizualizării informațiilor referitoare la clienții stocați în baza de date, o a doua fereastră pentru vizualizarea, inserarea și editarea produselor înregistrate și o a treia fereastră pentru înregistrarea comenzilor și generarea facturilor pe baza acestora.

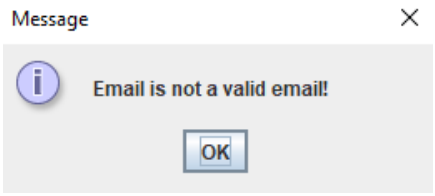
Cele trei ferestre dezvoltate cu scopul introducerii datelor de către utilizator au un panou principal în care au fost adăugate diferite subpanouri, implementate folosind JPanel. Astfel, interfața grafică are în componența sa câmpuri de text editabile pentru introducerea informațiilor în vederea înregistrării a unui nou client sau produs de către utilizator. Etichetele sunt realizate folosind JLabel, în timp ce câmpurile de text au la bază componente de tip JTextField. De asemenea, în interfața grafică a aplicației se pot regăsi numeroase butoane butoane, realizate folosind componente JButton. Tabelele ce conțin clienții și produsele au fost plasate în panouri de derulare de tip JScrollPanel pentru a permite vizualizarea acestora în întregime. Componentele interfeței grafice pot fi vizualizate în figurile de mai jos:





Aplicația semnalizează o eroare prin apariția unei ferestre cu un mesaj specific în cazul în care utilizatorul introduce date invalide. Ferestrele ce afișează mesajele de eroare au următorul design:

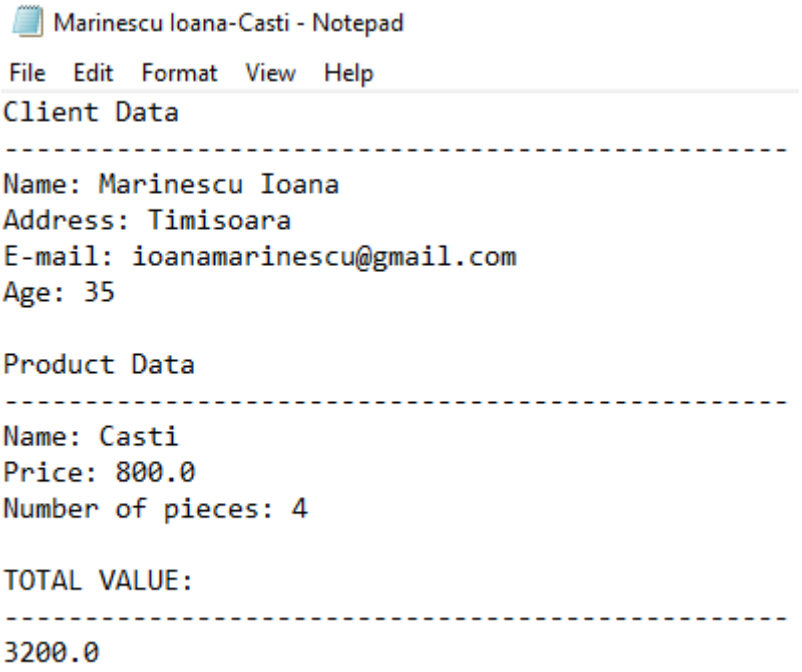




Interfața grafică facilitează folosirea simulatorului de către orice tip de utilizator și oferă o interacțiune mai bună a utilizatorului modelul obiectual al soluției.

5. Rezultate

Testarea aplicației a fost realizată astfel încât să acopere toate cazurile de utilizare, atât în scenariul de succes, cât și în cel de eșec. Aplicația a reacționat corect, având comportamentul așteptat în toate cazurile de testare furnizate, a respectat cadrul de cerințe funcționale impus prin specificația problemei, în fișierul .txt generat putând fi regăsite pentru fiecare comandă informațiile referitoare la clientul care a plasat comanda, informațiile referitoare la produsul comandat și totalul de plată, după cum se poate observa din figura de mai jos:



Testarea proiectului s-a realizat pe un set larg de teste, aplicația având un răspuns pozitiv la multitudinea de cazuri de test furnizate, printre acestea numărându-se, de asemenea, situații de tip excepție care au fost interceptate și tratate corect. De asemenea, la fiecare pas de testare s-a verificat concordanța rezultatelor scrise în fișierul ce conține factura atașată comenzii cu cele înregistrate în baza de date.

6. Concluzii

Concluzionând, se poate afirma că această temă vine să soluționeze problema ineficienței înregistrării comenzilor și a consistenței datelor în aplicațiile de marketing construite pe scheletul unei baze de date, fiind implementată conform paradigmei programării orientate pe obiect și proiectându-se ca o mini-aplicație ce poate fi folosită de orice tip de utilizator, dispunând de un pattern arhitectural care permite separarea modelului obiectual, de interfața grafică cu care utilizatorul interacționează, fiecare nivel al logicii interne fiind bine delimitat. Pentru dezvoltatorul proiectului, implementarea acestuia s-a dovedit utilă, având drept rezultat o mai bună însușire a manipulării și dezvoltării aplicațiilor Object Oriented, prin exersarea implementării practice a conceptelor specifice acestei paradigme de programare, dar și acumularea de noi cunoștințe referitoare realizarea conexiunii unui proiect dezvoltat în Java la o bază de date, a tipurilor de date și obiecte abstracte și a elementelor de grafică și vizualizare în dezvoltarea unei aplicații susținute de o interfață grafică cu utilizatorul.

Printre posibilele dezvoltări ulterioare ale aplicației se numără afișarea unei ferestre ce conține un mesaj de confirmare la înregistrarea cu succes a comenzii, dar și posibilitatea de a renunța la comanda plasată într-un anumit interval de timp de la înregistrarea acesteia.

7. Bibliografie

- ASSIGNMENT_3_SUPPORT_PRESENTATION.pdf (furnizat pe Microsoft Teams)
- Tema3.pdf (furnizat pe Microsoft Teams)
- Cursuri și lucrări de laborator de la disciplina Programare Orientată pe Obiecte, studiată în semestrul I al anului universitar 2020-2021
- https://mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/
- http://tutorials.jenkov.com/java-reflection/index.html
- https://www.baeldung.com/javadoc
- https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html