Author: Georgescu Vlad
Group: 30424

Documentation
Programming Techniques, Homework 3
Order Management application

Content

Author: Georgescu Vlad
Group: 30424

<u>1. Homework's Objective</u>

The aim of this homework and its principal objective is to design an application for processing customer orders for a warehouse.

The secondary objectives need to be accomplished in order to fulfill the main objective. The secondary objectives are:

Secondary Objectives	**Description**
1. Identifying the real life problem's possible scenarios[1]	This secondary objective is mandatory in the flow of developing the system in order to identify the system's requirements for designing them afterwise
2. Decisions regarding structure of application[2]	Designing the needed classes and their specific functionality
3. Graphical User Interface[3]	Creating a nice and simple Graphical User Interface such that the system is accessible, interactive and easy to be understood by any user
4. Assembling the components together [4]	Linking the classes and the graphical user interface together in order to complete the system's functionality
5. Testing the application[5]	The results of application should be analysed in order to determine if the system is having a correct functionality or not.

<u>2. Problem analysis, modelling</u>

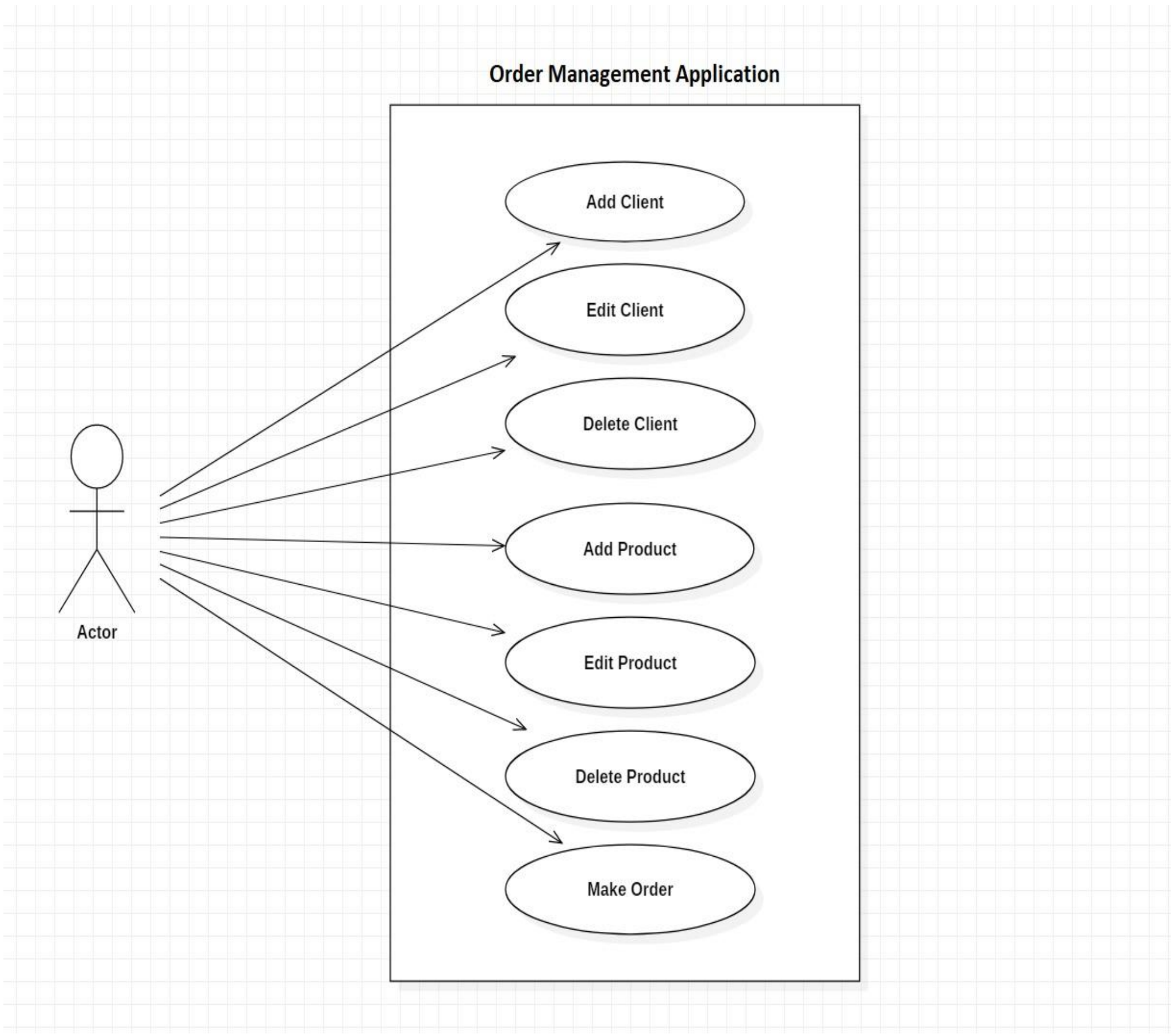
The system performs orders for a warehouse. Clients and Products can be added, removed or edited and orders can be generated. Orders can be generated using clients and products and inserting the desired quantity.

In order to present how the system is used, an *use-case diagram* that presents a set of scenarios that describe the "natural" behavior of the system is being created.

Author: Georgescu Vlad
Group: 30424

The scenarios are the use cases which in the current case are the operations that can be done on the products and clients and the generation of an order.

Use-case diagram:



Use-cases description:

1. Add Client
- User inserts the client properties: name, address, city, email, telephone number
- System collects the data and checks to see if all the fields were inserted correctly
- System displays a message if the email or telephone number were not inserted correctly and returns the control back to the user to insert again the client
- System inserts the client into the database and into the JTable if the inserted data was correct

2. Edit Client
- User inserts the client properties: name, address, city, email, telephone number
- System collects the data and checks to see if all the fields were inserted correctly
- System displays a message if the email or telephone number were not inserted correctly and returns the control back to the user to insert again the client
- System updates the client into the database and into the JTable if the inserted data was correct

3. Delete Client
- User inserts the name of the client that he/she wants to delete
- System collects the data and checks to see if the name was inserted correctly
- System displays a message if it was not a valid name and returns the control back to the user to insert again the name of the client
- System deletes the client from the database and from the JTable if the inserted name was correct

4. Add Product
- User inserts the product properties: name, quantity, price
- System collects the data and checks to see if all the fields were inserted correctly
- System displays a message if the name was not inserted correctly and returns the control back to the user to insert again the product

- System inserts the product into the database and into the JTable if the inserted data was correct

5. Edit Product
- User inserts the product properties: name, quantity, price
- System collects the data and checks to see if all the fields were inserted correctly
- System displays a message if the name of the product was not inserted correctly and returns the control back to the user to insert again the product
- System updates the product into the database and into the JTable if the inserted data was correct

6. Delete Product
- User inserts the name of the product that he/she wants to delete
- System collects the data and checks to see if the name was inserted correctly
- System displays a message if it was not a valid name and returns the control back to the user to insert again the name of the product
- System deletes the product from the database and from the JTable if the inserted name was correct

7. Make Order
- User inserts selects the client from the table of clients, selects the product from the table of products and inserts the desired quantity
- System collects the data and checks to see if there is enough stock available for the product selected
- System displays a message if the stock available is less than the quantity selected by the user and returns the control back to the user to insert again the quantity of the desired product
- System generates the order into order table from the database if the inserted data was correct and creates an Order Bill saved as txt

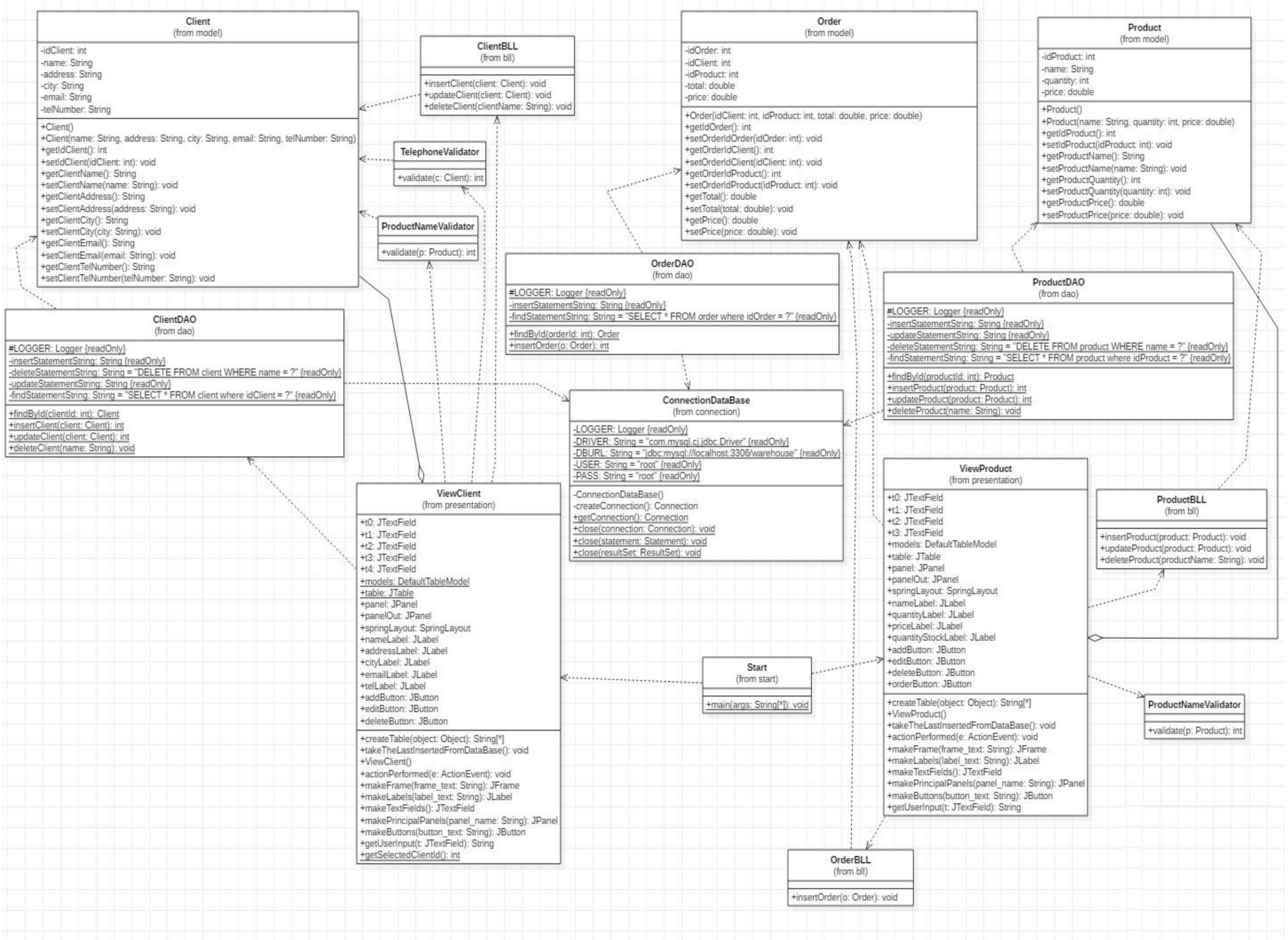
3. Design

OOP designing of application:

The application was designed for processing customer orders for a warehouse. Products and Clients features must be inserted from the user interface into database. They can be also edited or deleted after being inserted.

As data structures used, there was used an ArrayList of strings for storing the name of the colums of the JTable whose header was generated through reflexion technique.

The Class Diagram created using StarUML of the system is the following[6]:


Client
(from model)
-idClient: int
-name: String
-address: String
-city: String
-email: String
-telNumber: String
+Client()
+Client(name: String, address: String, city: String, email: String, telNumber: String)
+getIdClient(): int
+setIdClient(idClient: int): void
+getClientName(): String
+setClientName(name: String): void
+getClientAddress(): String
+setClientAddress(address: String): void
+getClientCity(): String
+setClientCity(city: String): void
+getClientEmail(): String
+setClientEmail(email: String): void
+getClientTelNumber(): String
+setClientTelNumber(telNumber: String): void
ClientBLL
(from bll)
+insertClient(client: Client): void
+updateClient(client: Client): void
+deleteClient(clientName: String): void
TelephoneValidator
+validate(c: Client): int
ProductNameValidator
+validate(p: Product): int
Order
(from model)
-idOrder: int
-idClient: int
-idProduct: int
-total: double
-price: double
+Order(idClient: int, idProduct: int, total: double, price: double)
+getIdOrder(): int
+setOrderIdOrder(idOrder: int): void
+getOrderIdClient(): int
+setOrderIdClient(idClient: int): void
+getOrderIdProduct(): int
+setOrderIdProduct(idProduct: int): void
+getTotal(): double
+setTotal(total: double): void
+getPrice(): double
+setPrice(price: double): void
Product
(from model)
-idProduct: int
-name: String
-quantity: int
-price: double
+Product()
+Product(name: String, quantity: int, price: double)
+getIdProduct(): int
+setIdProduct(idProduct: int): void
+getProductName(): String
+setProductName(name: String): void
+getProductQuantity(): int
+setProductQuantity(quantity: int): void
+getProductPrice(): double
+setProductPrice(price: double): void
ClientDAO
(from dao)
#LOGGER: Logger {readOnly}
-insertStatementString: String {readOnly}
-deleteStatementString: String = "DELETE FROM client WHERE name = ?" {readOnly}
-updateStatementString: String {readOnly}
-findStatementString: String = "SELECT * FROM client where idClient = ?" {readOnly}
+findById(clientId: int): Client
+insertClient(client: Client): int
+updateClient(client: Client): int
+deleteClient(name: String): void
OrderDAO
(from dao)
#LOGGER: Logger {readOnly}
-insertStatementString: String {readOnly}
-findStatementString: String = "SELECT * FROM order where idOrder = ?" {readOnly}
+findById(orderId: int): Order
+insertOrder(o: Order): int
ProductDAO
(from dao)
#LOGGER: Logger {readOnly}
-insertStatementString: String {readOnly}
-updateStatementString: String {readOnly}
-deleteStatementString: String = "DELETE FROM product WHERE name = ?" {readOnly}
-findStatementString: String = "SELECT * FROM product where idProduct = ?" {readOnly}
+findById(productId: int): Product
+insertProduct(product: Product): int
+updateProduct(product: Product): int
+deleteProduct(name: String): void
ConnectionDataBase
(from connection)
-LOGGER: Logger {readOnly}
-DRIVER: String = "com.mysql.cj.jdbc.Driver" {readOnly}
-DBURL: String = "jdbc:mysql://localhost:3306/warehouse" {readOnly}
-USER: String = "root" {readOnly}
-PASS: String = "root" {readOnly}
-ConnectionDataBase()
-createConnection(): Connection
+getConnection(): Connection
+close(connection: Connection): void
+close(statement: Statement): void
+close(resultSet: ResultSet): void
ViewClient
(from presentation)
+t0: JTextField
+t1: JTextField
+t2: JTextField
+t3: JTextField
+t4: JTextField
+models: DefaultTableModel
+table: JTable
+panel: JPanel
+panelOut: JPanel
+springLayout: SpringLayout
+nameLabel: JLabel
+addressLabel: JLabel
+cityLabel: JLabel
+emailLabel: JLabel
+telLabel: JLabel
+addButton: JButton
+editButton: JButton
+deleteButton: JButton
+createTable(object: Object): String[*]
+takeTheLastInsertedFromDataBase(): void
+ViewClient()
+actionPerformed(e: ActionEvent): void
+makeFrame(frame_text: String): JFrame
+makeLabels(label_text: String): JLabel
+makeTextFields(): JTextField
+makePrincipalPanels(panel_name: String): JPanel
+makeButtons(button_text: String): JButton
+getUserInput(t: JTextField): String
+getSelectedClientId(): int
ViewProduct
(from presentation)
+t0: JTextField
+t1: JTextField
+t2: JTextField
+t3: JTextField
+models: DefaultTableModel
+table: JTable
+panel: JPanel
+panelOut: JPanel
+springLayout: SpringLayout
+nameLabel: JLabel
+quantityLabel: JLabel
+priceLabel: JLabel
+quantityStockLabel: JLabel
+addButton: JButton
+editButton: JButton
+deleteButton: JButton
+orderButton: JButton
+createTable(object: Object): String[*]
+ViewProduct()
+takeTheLastInsertedFromDataBase(): void
+actionPerformed(e: ActionEvent): void
+makeFrame(frame_text: String): JFrame
+makeLabels(label_text: String): JLabel
+makeTextFields(): JTextField
+makePrincipalPanels(panel_name: String): JPanel
+makeButtons(button_text: String): JButton
+getUserInput(t: JTextField): String
ProductBLL
(from bll)
+insertProduct(product: Product): void
+updateProduct(product: Product): void
+deleteProduct(productName: String): void
Start
(from start)
+main(args: String[*]): void
ProductNameValidator
+validate(p: Product): int
OrderBLL
(from bll)
+insertOrder(o: Order): void

There are the following relationships between the classes of the application:
- We can distinguish the following dependency relationships between the following classes: ClientDAO and Client, OrderDAO and Order, ProductDAO and Product, ClientDAO and ConnectionDataBase, ProductDAO and ConnectionDataBase, TelephoneValidator and Client, ProductNameValidator and Client, ViewClient and TelephoneValidator, ViewClient and ProductNameValidator, ViewClient and ClientBLL, ViewProduct and ProductBLL, ViewProduct and ProductNameValidator, ClientBLL and Client, OrderBLL and Order, ProductBLL and Product, Start and ViewClient, Start and ViewProduct because the dependent class uses an object of type of the class which it depends on. The dependent class is the one written in the left of the "and" word above.
- There is an aggregation relationship between Product and ViewProduct because both entities can survive individually but ViewProduct has/uses many Products.
- There is an aggregation relationship between Client and ViewClient because both entities can survive individually but ViewClient has/uses many Clients.

4. Implementation

The application is structured in 7 packages, using layered architecture, and 16 classes.

The package presentation contains the classes that defines the user interface. It also contains the action listeners for the buttons that user uses to interact with the application.

Class ViewClient constructs the window related to client features and operations. It uses a grid layout for the window to separate the JTable[7] from the buttons and textfields. It contains JButtons for adding, editing and removing a client, JTextFields[8] for entering the client's name, address, city, email, and telephone number. In the right hand side, there is a JTable containing all the clients inserted in the database along with all their features. The head of the JTable is generated through reflexion technique using the method createTable() which gets the fields of the class Client through reflexion[9] and insert them as strings into an array list of strings.

The class ViewClient implements also the action listeners on the buttons. When user press the Add Client button, the application retrieves the data about client and insert it into the database if the data inserted was correct. Then, the data from the database is taken and the JTable from the user interface is updated. In order to take the latest added client from the database and to insert it into the JTable, the method takeTheLastInsertedFromDataBase() is used. When Edit Client button is pressed, there is the same process as in case of adding at new client, but with the only difference that now there is not added a new client but updated an existing one. When Delete Button is pressed, the client with the name inserted into the text field corresponding to the name is deleted from the database and then the JTable is updated.

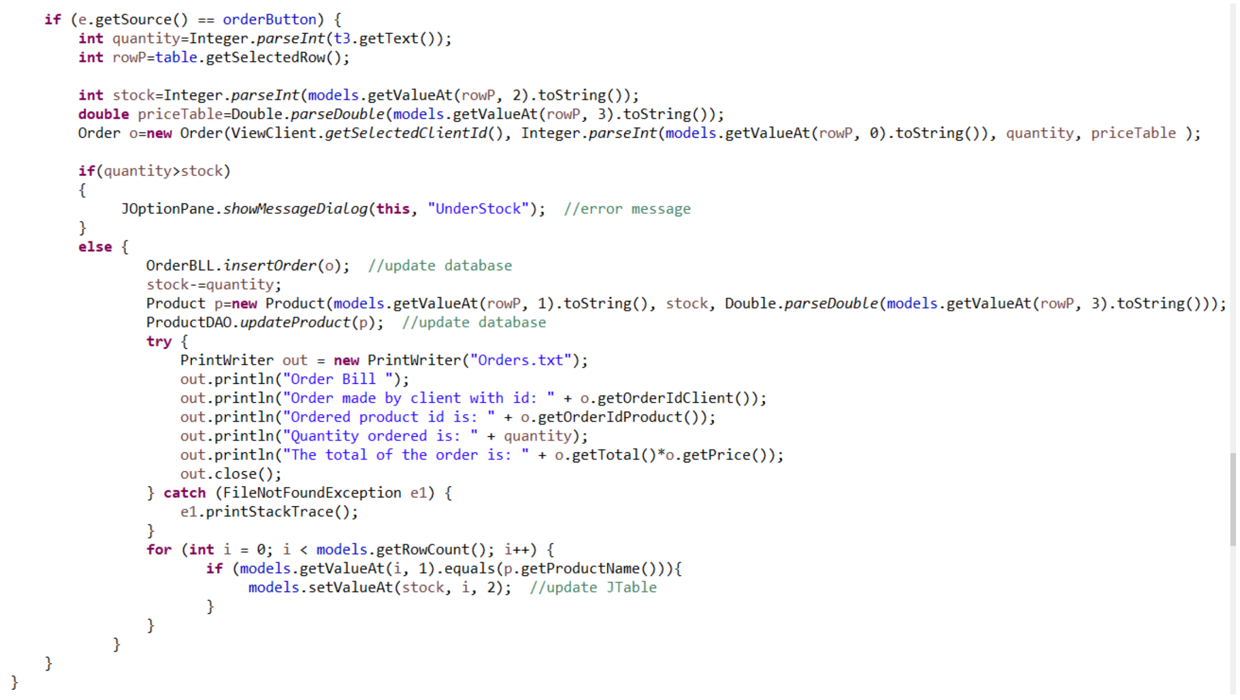
There are also methods for creating the UI components and one method called getSelectedClientId() which returns the id of the selected client from the JTable of clients. This method is used in ViewProduct class when the user wants to create an order.

Class ViewProduct constructs the window related to product features and operations. It uses a grid layout for the window to separate the JTable from the buttons and text fields. There are also methods for creating the UI components. It contains JButtons for adding, editing and removing a product, JText fields for entering the product's name, quantity and price. In the right hand side, there is a JTable containing all the products inserted in the database along with all their features. The head of the JTable is generated through reflexion technique using the method createTable() which gets the fields of the class Product through reflexion and insert them as strings into an array list of strings.

```
public ArrayList<String> createTable(Object object)
{
    ArrayList<String> str=new ArrayList<String>();
    Class cls = object.getClass();
    try {
        Field[] field = cls.getDeclaredFields();
        for(int i=0; i<field.length; i++)
        {
            str.add(field[i].getName());
        }
    } catch (SecurityException e) {
        e.printStackTrace();
    }
    return str;
}
```

The class ViewProduct implements also the action listeners on the buttons. When user press the Add Product button, the application retrieves the data about product and insert it into the database if the data inserted was correct. Then, the data from the database is taken and the JTable from the user interface is updated. In order to take the latest added product from the database and to insert it into the JTable, the method takeTheLastInsertedFromDataBase() is used. When Edit Product button is pressed, there is the same process as in case of adding at new product, but with the only difference that now there is not added a new product but updated an existing one. When Delete Button is pressed, the product with the name inserted into the text field corresponding to the name is deleted from the database and then the JTable is updated.

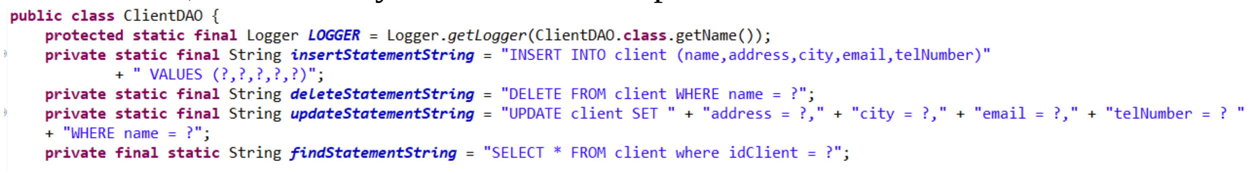
When Make Order button is pressed, the quantity inserted into its corresponding text field is taken and if it is smaller or equal with the selected product's quantity, an order with the id of the selected client and selected product and with the total computed by multiplying the quantity desired with the price of the selected product is being generated into Order table of the database and also an Order Bill saved as txt with name Orders.txt. Otherwise, a popup message indicating the under stock for the selected product is being displayed.



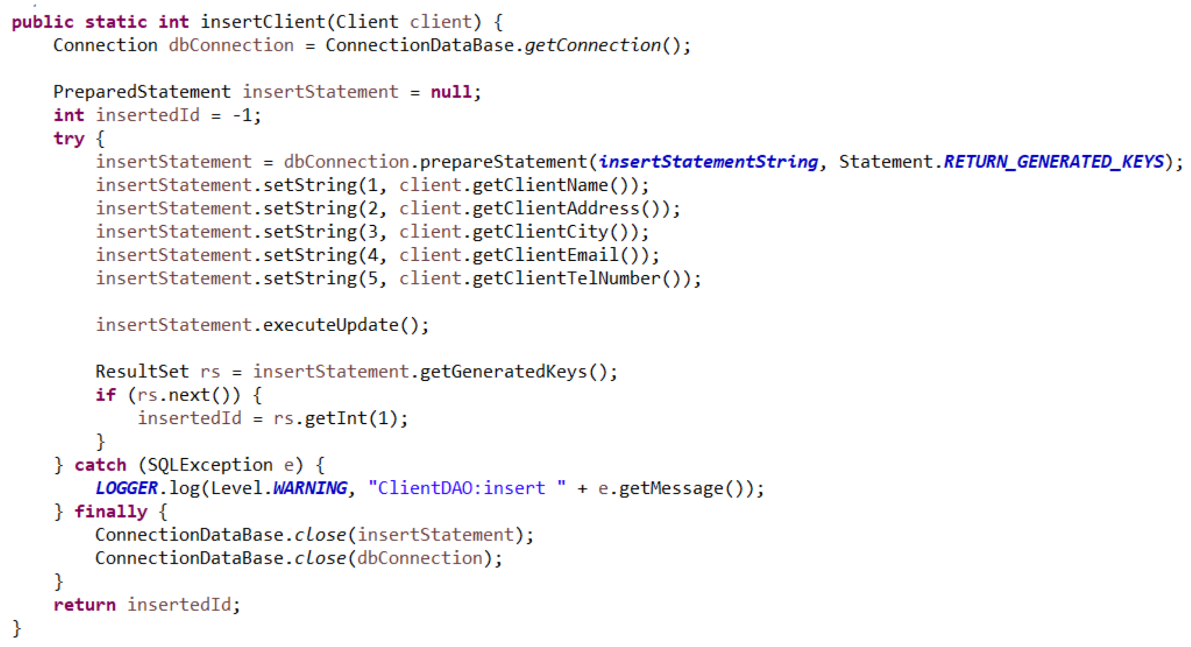
```
    if (e.getSource() == orderButton) {
        int quantity=Integer.parseInt(t3.getText());
        int rowP=table.getSelectedRow();

        int stock=Integer.parseInt(models.getValueAt(rowP, 2).toString());
        double priceTable=Double.parseDouble(models.getValueAt(rowP, 3).toString());
        Order o=new Order(ViewClient.getSelectedClientId(), Integer.parseInt(models.getValueAt(rowP, 0).toString()), quantity, priceTable );

        if(quantity>stock)
        {
            JOptionPane.showMessageDialog(this, "UnderStock");  //error message
        }
        else {
                OrderBLL.insertOrder(o);  //update database
                stock-=quantity;
                Product p=new Product(models.getValueAt(rowP, 1).toString(), stock, Double.parseDouble(models.getValueAt(rowP, 3).toString()));
                ProductDAO.updateProduct(p);  //update database
                try {
                    PrintWriter out = new PrintWriter("Orders.txt");
                    out.println("Order Bill ");
                    out.println("Order made by client with id: " + o.getOrderIdClient());
                    out.println("Ordered product id is: " + o.getOrderIdProduct());
                    out.println("Quantity ordered is: " + quantity);
                    out.println("The total of the order is: " + o.getTotal()*o.getPrice());
                    out.close();
                } catch (FileNotFoundException e1) {
                    e1.printStackTrace();
                }
                for (int i = 0; i < models.getRowCount(); i++) {
                        if (models.getValueAt(i, 1).equals(p.getProductName())){
                            models.setValueAt(stock, i, 2);  //update JTable
                        }
                }
            }
    }
}
```

Package DAO contains 3 classes responsible for querries and connection to the database.

Class ClientDAO contains sql querries to insert, update and delete a client from the database and it has 3 methods for these operations that uses those querries. The method insertClient inserts the client received as parameter into the database in the table client. The method updateClient updates the client received as parameter into the database. The method deleteClient deletes the client received as parameter into the database. It has also a method findById() which finds a client by its id into the database, but it is only for future developments.



```
public class ClientDAO {
    protected static final Logger LOGGER = Logger.getLogger(ClientDAO.class.getName());
    private static final String insertStatementString = "INSERT INTO client (name,address,city,email,telNumber)"
            + " VALUES (?,?,?,?,?)";
    private static final String deleteStatementString = "DELETE FROM client WHERE name = ?";
    private static final String updateStatementString = "UPDATE client SET " + "address = ?," + "city = ?," + "email = ?," + "telNumber = ? "
    + "WHERE name = ?";
    private final static String findStatementString = "SELECT * FROM client where idClient = ?";
```



```
    public static int insertClient(Client client) {
        Connection dbConnection = ConnectionDataBase.getConnection();

        PreparedStatement insertStatement = null;
        int insertedId = -1;
        try {
            insertStatement = dbConnection.prepareStatement(insertStatementString, Statement.RETURN_GENERATED_KEYS);
            insertStatement.setString(1, client.getClientName());
            insertStatement.setString(2, client.getClientAddress());
            insertStatement.setString(3, client.getClientCity());
            insertStatement.setString(4, client.getClientEmail());
            insertStatement.setString(5, client.getClientTelNumber());

            insertStatement.executeUpdate();

            ResultSet rs = insertStatement.getGeneratedKeys();
            if (rs.next()) {
                insertedId = rs.getInt(1);
            }
        } catch (SQLException e) {
            LOGGER.log(Level.WARNING, "ClientDAO:insert " + e.getMessage());
        } finally {
            ConnectionDataBase.close(insertStatement);
            ConnectionDataBase.close(dbConnection);
        }
        return insertedId;
    }
```

Class ProductDAO contains sql querries to insert, update and delete a product from the database and it has 3 methods for these operations that uses those querries. The method insertProduct inserts the product received as parameter into the database in the table product. The method updateProduct updates the product received as parameter into the database. The method deleteProduct deletes the product received as parameter into the database. It has also a method findById() which finds a product by its id into the database, but it is only for future developments.

Class OrderDAO contains sql querry to insert an order into database. It has also a method insertOrder which inserts the order received as parameter into the database into order table. It has also a method findById() which finds an order by its id into the database, but it is only for future developments.

Package bll contains 3 classes that encapsulates the application logic. Class ClientBLL uses the methods insertClient(), updateClient() and deleteClient() that call the methods insertClient(), updateClient() and deleteClient() from class ClientDAO to perform the related operations to database. Class ProductBLL uses the methods insertProduct(), updateProduct() and deleteProduct() that call the methods insertProduct(), updateProduct() and deleteProduct() from class ProductDAO to perform the related operations to database. Class OrderBLL uses the method insertOrder() from class OrderDAO to insert an ordr into the database.

Package bll.validators contains 3 classes that performs the validations. Class EmailValidator uses the method validate() that checks to see if the inserted email of the specific client was correct. The email is checked to contain at least three characters, to have only one @ and not many or less, and that the email does not start or end with an @. Class TelephoneValidator uses the method validate() that checks to see if the inserted telephone number of the specific client was correct. The telephone number is checked to contain only digits and to have exactly 10 digits. Class ProductNameValidator uses the method validate() that checks to see if the inserted name of the product is a valid name. The name is checked to contain only letters.

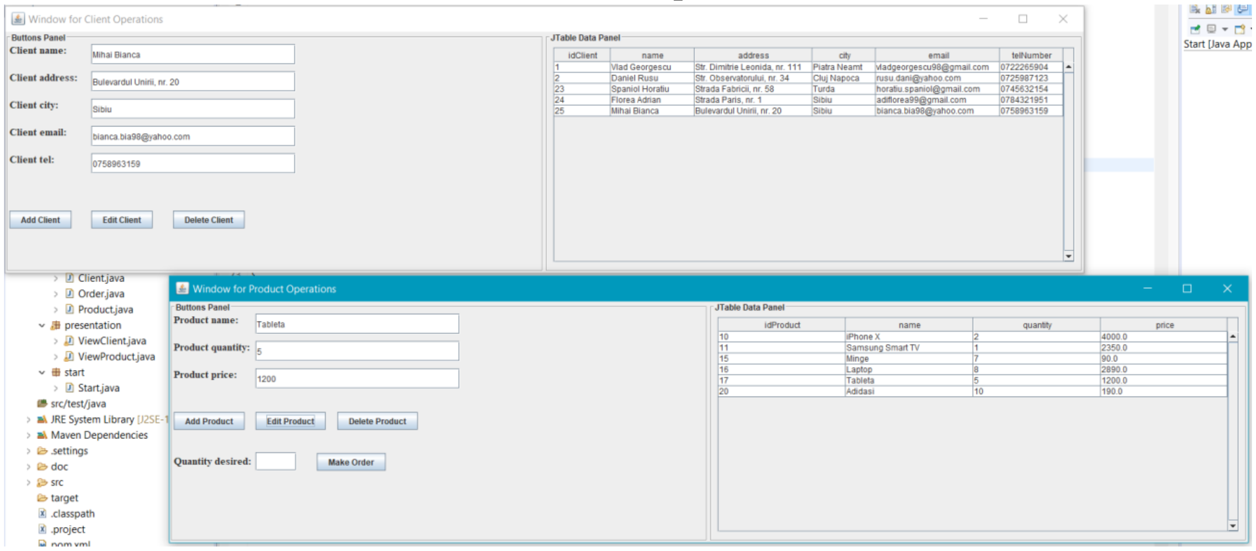
Package connection contains the ConnectionDataBase class which realises the connection to the database. It has a method createConnection() to create the connection to the database, a method getConnection() to get the connection to database and three close methods to close the connection, statement and result.

Package model contains three classes: Client, Product, and Order. These classes contains the attributes for client, product and order and also setters and getters for these attributes.

Package start contains the Start class which is the program driver class. It has only one method, the main() method that calls the constructors of classes ViewClient and ViewProduct in order to set up the windows and start the application.

5. Results

In order to see the results, operations were made. We can observe below the addition of the new client and the edit of the product “tableta”



Client name: Mihai Bianca
Client address: Bulevardul Unirii, nr. 20
Client city: Sibiu
Client email: bianca.bia98@yahoo.com
Client tel: 0758963159

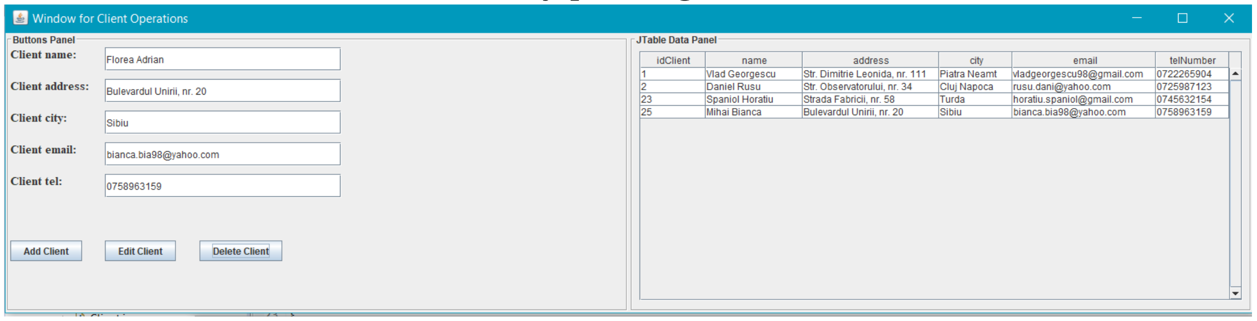
idClient	name	address	city	email	telNumber
1	Vlad Georgescu	Str. Dimitrie Leonida, nr. 111	Piatra Neamt	vladgeorgescu98@gmail.com	0722265904
2	Daniel Rusu	Str. Observatorului, nr. 34	Cluj Napoca	rusu.dani@yahoo.com	0725987123
23	Spaniol Horatiu	Strada Fabricii, nr. 58	Turda	horatiu.spaniol@gmail.com	0745632154
24	Florea Adrian	Strada Paris, nr. 1	Sibiu	adiflorea99@gmail.com	0784321951
25	Mihai Bianca	Bulevardul Unirii, nr. 20	Sibiu	bianca.bia98@yahoo.com	0758963159

Client.java
Order.java
Product.java
presentation
ViewClient.java
ViewProduct.java
start
Start.java
src/test/java
JRE System Library [J2SE-1
Maven Dependencies
.settings
doc
src
target
.classpath
.project

Product name: Tableta
Product quantity: 5
Product price: 1200
Quantity desired:

idProduct	name	quantity	price
10	iPhone X	2	4000.0
11	Samsung Smart TV	1	2350.0
15	Minge	7	90.0
16	Laptop	8	2890.0
17	Tableta	5	1200.0
20	Adidasi	10	190.0

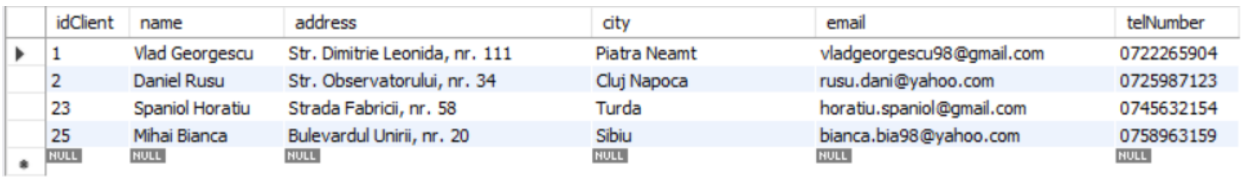
Let observe now a deletion: We deleted the client “Florea Adrian” by entering his name into Client Name text field and by pressing the delete client button.



Client name: Florea Adrian
Client address: Bulevardul Unirii, nr. 20
Client city: Sibiu
Client email: bianca.bia98@yahoo.com
Client tel: 0758963159

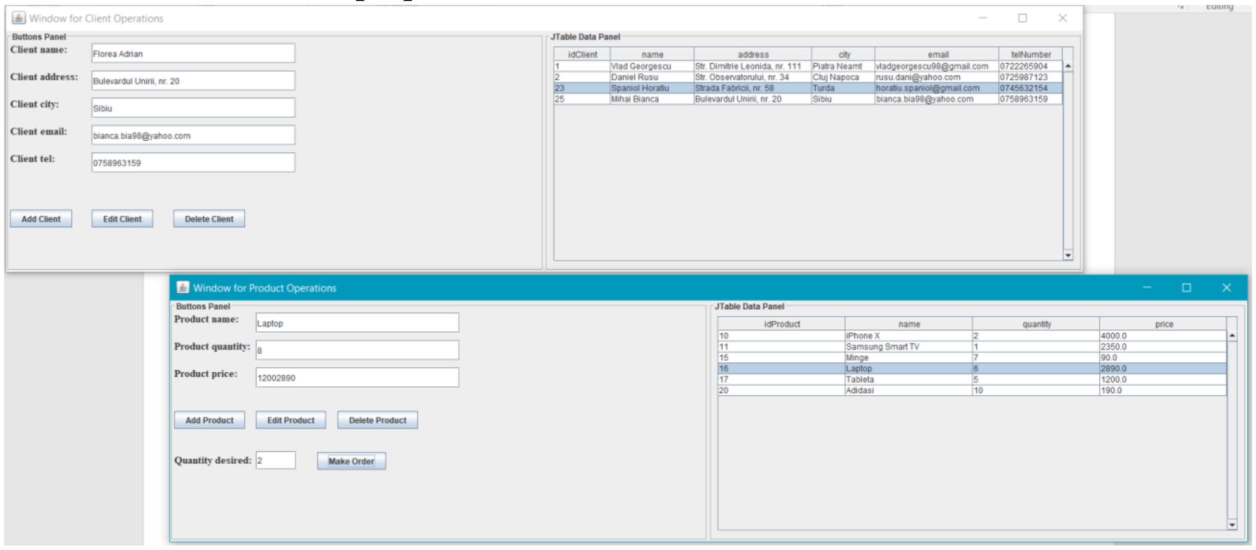
idClient	name	address	city	email	telNumber
1	Vlad Georgescu	Str. Dimitrie Leonida, nr. 111	Piatra Neamt	vladgeorgescu98@gmail.com	0722265904
2	Daniel Rusu	Str. Observatorului, nr. 34	Cluj Napoca	rusu.dani@yahoo.com	0725987123
23	Spaniol Horatiu	Strada Fabricii, nr. 58	Turda	horatiu.spaniol@gmail.com	0745632154
25	Mihai Bianca	Bulevardul Unirii, nr. 20	Sibiu	bianca.bia98@yahoo.com	0758963159

Let us observe now the changes into the database:



	idClient	name	address	city	email	telNumber
▶	1	Vlad Georgescu	Str. Dimitrie Leonida, nr. 111	Piatra Neamt	vladgeorgescu98@gmail.com	0722265904
	2	Daniel Rusu	Str. Observatorului, nr. 34	Cluj Napoca	rusu.dani@yahoo.com	0725987123
	23	Spaniol Horatiu	Strada Fabricii, nr. 58	Turda	horatiu.spaniol@gmail.com	0745632154
	25	Mihai Bianca	Bulevardul Unirii, nr. 20	Sibiu	bianca.bia98@yahoo.com	0758963159
*	NULL	NULL	NULL	NULL	NULL	NULL

Let us now realise an order. Let suppose that the client "Spaniol Horatiu" wants to buy 2 laptops. The stock of laptops before order was 8 and the price for 1 laptop is 2890. The number of laptops are decremented to 6.



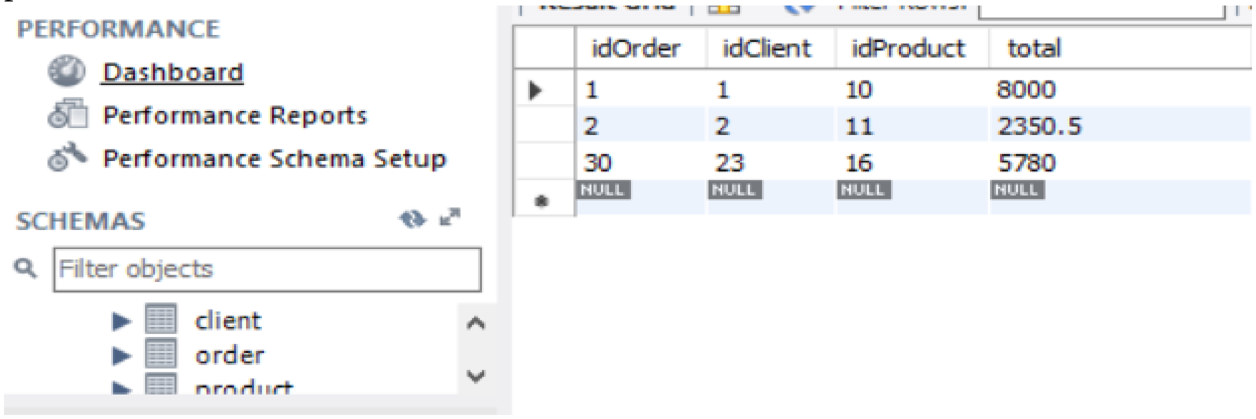
Client name: Florea Adrian
Client address: Bulevardul Unirii, nr. 20
Client city: Sibiu
Client email: bianca.bia98@yahoo.com
Client tel: 0758963159

idClient	name	address	city	email	telNumber
1	Vlad Georgescu	Str. Dimitrie Leonida, nr. 111	Piatra Neamt	vladgeorgescu98@gmail.com	0722265904
2	Daniel Rusu	Str. Observatorului, nr. 34	Cluj Napoca	rusu.dani@yahoo.com	0725987123
23	Spaniol Horatiu	Strada Fabricii, nr. 58	Turda	horatiu.spaniol@gmail.com	0745632154
25	Mihai Bianca	Bulevardul Unirii, nr. 20	Sibiu	bianca.bia98@yahoo.com	0758963159

Product name: Laptop
Product quantity: 8
Product price: 12002890
Quantity desired: 2

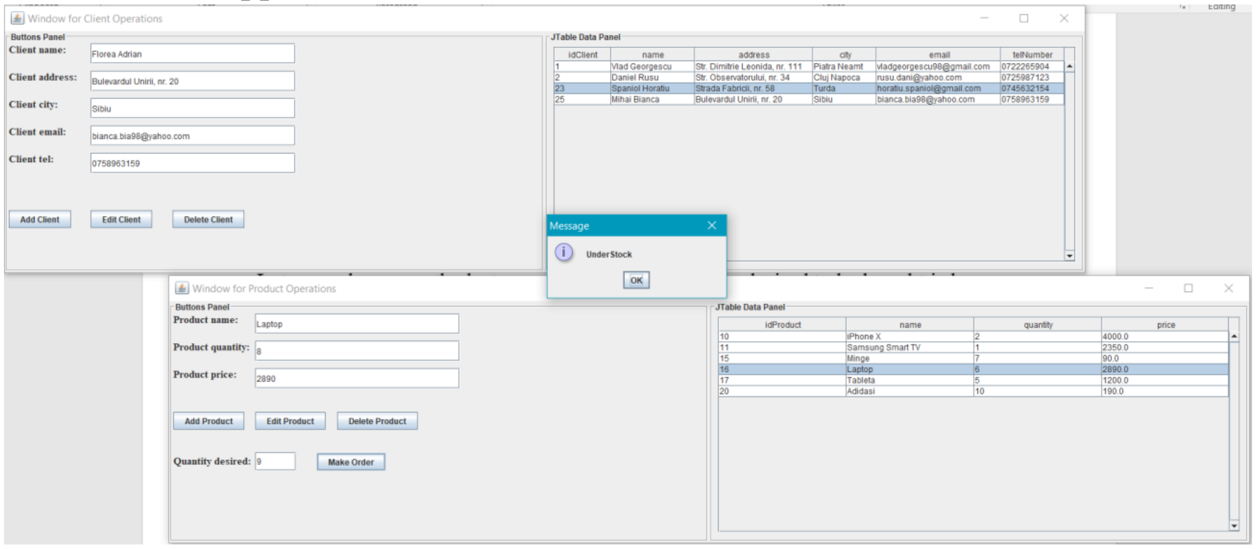
idProduct	name	quantity	price
10	iPhone X	2	4000.0
11	Samsung Smart TV	1	2350.0
15	Minge	7	90.0
16	Laptop	6	2890.0
17	Tableta	5	1200.0
20	Adidasi	10	190.0

And we can see in the database in the table order the new added order with the id of "Spaniol Horatiu" which is 23 and with the id of Laptop which is 16, and with total price of 2*2890=5780:



	idOrder	idClient	idProduct	total
▶	1	1	10	8000
	2	2	11	2350.5
	30	23	16	5780
*	NULL	NULL	NULL	NULL

Let us analyze now the last case, in which the quantity desired to be bought is larger than the available stock. Suppose now that the client "Spaniol Horatiu" wants to buy again laptops, but now he wants 9 instead of 2. The popup with the message "UnderStock" appeared:



Client name: Florea Adrian
Client address: Bulevardul Unirii, nr. 20
Client city: Sibiu
Client email: bianca.bia98@yahoo.com
Client tel: 0758963159

idClient	name	address	city	email	telNumber
1	Vlad Georgescu	Str. Dimitrie Leonida, nr. 111	Piatra Neamt	vladgeorgescu98@gmail.com	0722265904
2	Daniel Rusu	Str. Observatorului, nr. 34	Cluj Napoca	rusu.dani@yahoo.com	0725987123
23	Spaniol Horatiu	Strada Fabricii, nr. 58	Turda	horatiu.spaniol@gmail.com	0745632154
25	Mihai Bianca	Bulevardul Unirii, nr. 20	Sibiu	bianca.bia98@yahoo.com	0758963159

UnderStock

Product name: Laptop
Product quantity: 8
Product price: 2890
Quantity desired: 9

idProduct	name	quantity	price
10	iPhone X	2	4000.0
11	Samsung Smart TV	1	2350.0
15	Minge	7	90.0
16	Laptop	6	2890.0
17	Tableta	5	1200.0
20	Adidasi	10	190.0

6. Conclusions

To conclude, the system for order processing was very useful in accomplish new object oriented programming skills. I learnt how to establish a connection to a database, how to insert, edit and delete a record from database. Also, I learnt about JTable and how to use one. Also, the Reflexion technique was learnt and how to generate the head of a JTable uing ReflexionTechniques. Moreover, this documentation was helpful because I learnt how to structure and assembly all my ideas regarding the projection of the system all together.

As future developments, I would consider realizing an order with more than one product, using more than three tables in the database, that means the extension of the application, creation of a generic class that contains methods for accessing the database and implementing more validators.

7. Bibliography

- For creating the graphical user interface and link it to buttons:
https://docs.oracle.com/javase/8/docs/api/javax/swing/JFrame.html frame

https://docs.oracle.com/javase/8/docs/api/javax/swing/JButton.html buttons
https://www.javatpoint.com/java-jpanel panels
https://stackoverflow.com/questions/10177183/java-add-scroll-into-text-area scroll panel
https://docs.oracle.com/javase/8/docs/api/javax/swing/JTable.html JTable
https://docs.oracle.com/javase/tutorial/uiswing/components/textfield.html textfields
https://docs.oracle.com/javase/tutorial/uiswing/layout/spring.html layout
https://docs.oracle.com/javase/tutorial/uiswing/examples/components/TextDemoProject/src/components/TextDemo.java general gui example with all basic elements

- For UML Class Diagram
https://stackoverflow.com/questions/885937/what-is-the-difference-between-association-aggregation-and-composition/34069760#34069760
https://stackoverflow.com/questions/1230889/difference-between-association-and-dependency
https://stackoverflow.com/questions/21967841/aggregation-vs-composition-vs-association-vs-direct-association

- For application design
https://www.geeksforgeeks.org/reflection-in-java/ Reflexion technique
http://coned.utcluj.ro/~salomie/PT_Lic/
https://utcn_dsrl@bitbucket.org/utcn_dsrl/pt-layered-architecture.git Main source of information and code skeleton