DOCUMENTATION

ASSIGNMENT *ASSIGNMENT\_3*

STUDENT NAME: Tandea Sorin Darius

GROUP: 30422

Contents

[1. Main Objective: 3](#_Toc139244523)

[Sub-objectives: 3](#_Toc139244524)

[2. Analysis: 3](#_Toc139244525)

[Requirements: 3](#_Toc139244526)

[Functional Requirements: 3](#_Toc139244527)

[Non-Functional Requirements: 3](#_Toc139244528)

[Use Case Diagram: 4](#_Toc139244529)

[Use Case: 5](#_Toc139244530)

[3. Design 6](#_Toc139244531)

[3.1. OOP Design of the Application: 6](#_Toc139244532)

[3.2. UML Package and Class Diagrams: 6](#_Toc139244534)

[4. Implementation 8](#_Toc139244535)

[5. Results 14](#_Toc139244536)

[6. Conclusions 14](#_Toc139244537)

[7. Bibliography 14](#_Toc139244538)

# Main Objective:

Develop an Orders Management application for processing client orders in a warehouse using fundamental programming techniques. The application should be designed according to the layered architecture pattern and utilize relational databases for storing data.

## Sub-objectives:

* Analyze the problem and identify requirements
* Use object-oriented principles to design the application
* Use JavaDoc to document the classes
* Utilize relational databases to store the data for the application
* Use reflection techniques
* Implement the application using the layered architecture pattern
* Create an user intuitive gui

# Analysis:

## Requirements:

### Functional Requirements:

1. User Management:

* Allow users to create new client accounts.
* Provide functionality to edit and delete existing client accounts.

1. Product Management:

* Enable users to add new products to the database.
* Allow users to edit and delete existing products.
* Provide a view to display all products in a table format (JTable).

1. Order Management:

* Allow users to create product orders by selecting an existing product and client.
* Provide an interface to input the desired quantity for the product.
* Display an under-stock message if there are not enough products available.
* Decrement the product stock after the order is finalized.

1. GUI Display:

* Display all clients in a table format (JTable).
* Display all products in a table format (JTable).
* Display order details, including product, client, quantity, and total cost.

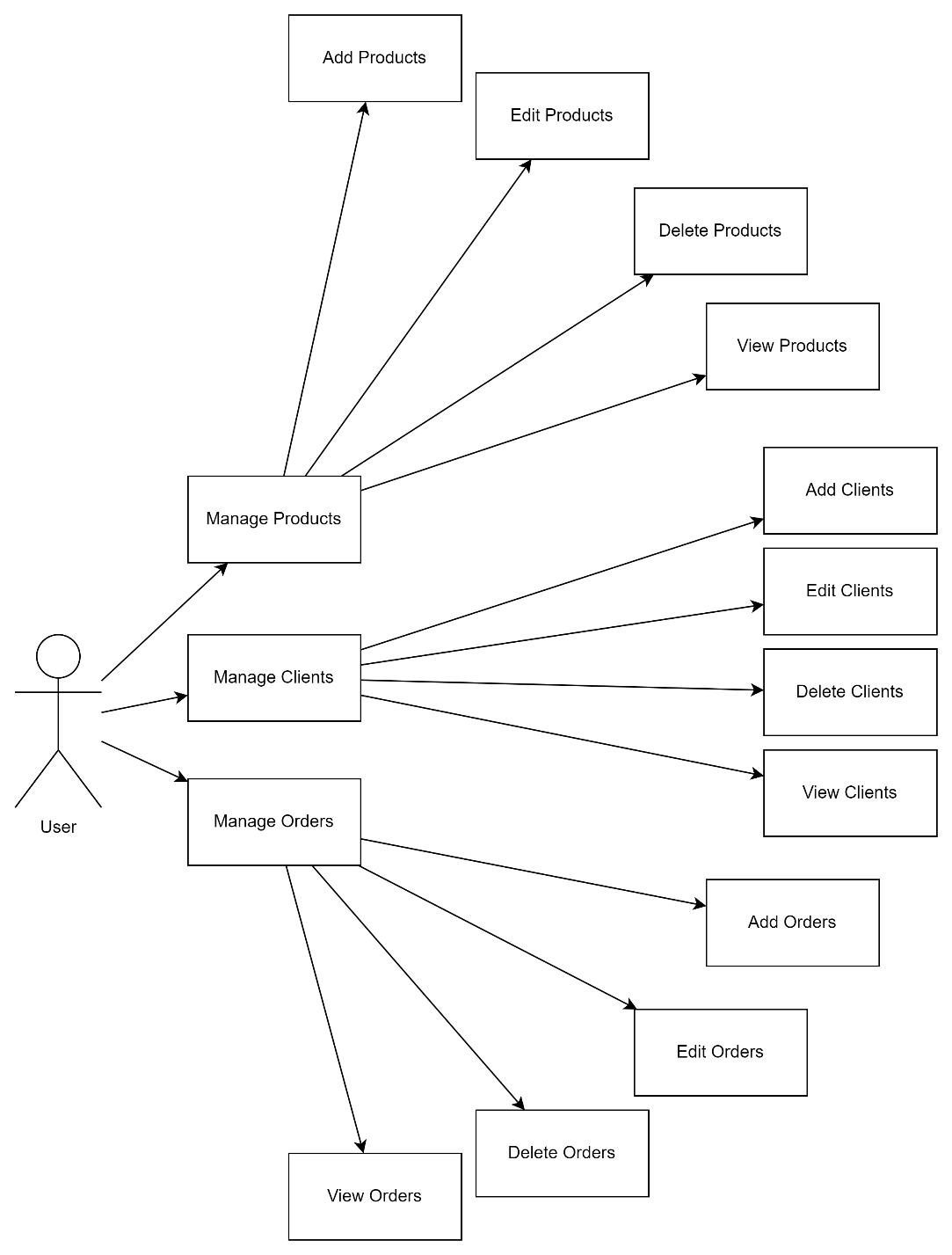
1. Reflection and Table Generation:

* Use reflection techniques to dynamically generate table headers based on object properties.
* Populate the tables with values from the object lists.

### Non-Functional Requirements:

* The GUI should be intuitive and user-friendly.
* Provide appropriate error messages and notifications to guide users.
* The application should handle exceptions and errors gracefully.
* Implement appropriate error handling and validation mechanisms.
* Follow coding conventions and best practices to ensure the code is clean and maintainable.
* Document the code adequately to facilitate future enhancements or modifications.
* The application should be designed to accommodate future growth and increasing data volume.
* Ensure that the database design and application architecture can scale with the business needs.
* The application should be developed in a way that it can run on different platforms without major modifications.

## Use Case Diagram:



## Use Case:

**Use Case**: Manage Clients

**Primary Actor**: User

**Precondition**: The user is logged into the system.

**Postcondition**: The client information is successfully managed (added, edited, deleted, or viewed).

**Steps**:

* User selects the "Manage Clients" option.
* User chooses to add a new client.
  + User enters the client details.
  + User submits the client information.
  + The system validates and saves the new client.
* User chooses to edit an existing client.
  + User selects a client to edit.
  + User modifies the client details.
  + User saves the changes.
* User chooses to delete a client.
  + User selects a client to delete.
  + User confirms the deletion.
  + The system removes the client from the database.
* User chooses to view all clients.
  + The system retrieves and displays a list of all clients.

**Use Case**: Manage Orders

**Primary** **Actor**: User

**Precondition**: The user is logged into the system and there are existing clients and products in the database.

**Postcondition**: The order is successfully managed (added, edited, deleted, or viewed).

**Steps**:

* User selects the "Manage Orders" option.
* User chooses to add a new order.
  + User selects a client for the order.
  + User selects products and specifies the quantities.
  + User submits the order.
  + The system validates the order and updates the product stock.
* User chooses to edit an existing order.
  + User selects an order to edit.
  + User modifies the order details (client, products, quantities).
  + User saves the changes.
* User chooses to delete an order.
  + User selects an order to delete.
  + User confirms the deletion.
  + The system removes the order and updates the product stock.
* User chooses to view all orders.
  + The system retrieves and displays a list of all orders.

**Use Case**: Manage Products

**Primary** **Actor**: User

**Precondition**: The user is logged into the system.

**Postcondition**: The product information is successfully managed (added, edited, deleted, or viewed).

**Steps**:

* User selects the "Manage Products" option.
* User chooses to add a new product.
  + User enters the product details.
  + User submits the product information.
  + The system validates and saves the new product.
* User chooses to edit an existing product.
  + User selects a product to edit.
  + User modifies the product details.
  + User saves the changes.
* User chooses to delete a product.
  + User selects a product to delete.
  + User confirms the deletion.
  + The system removes the product from the database.
* User chooses to view all products.
  + The system retrieves and displays a list of all products.

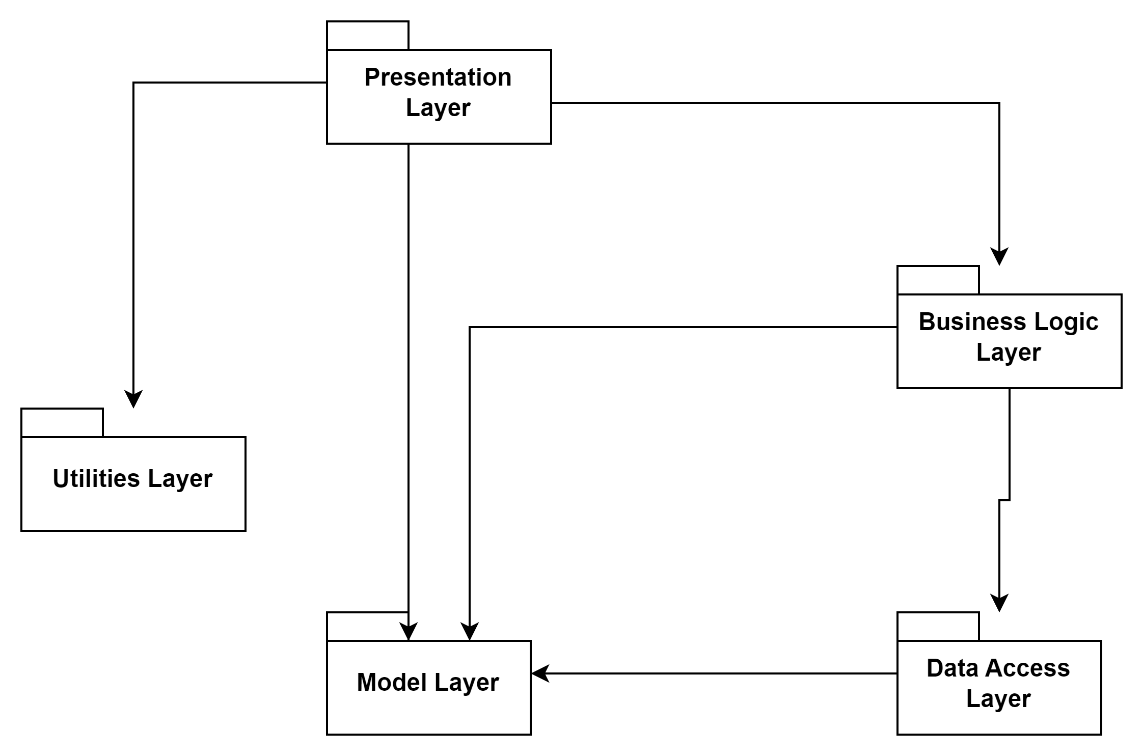
# Design

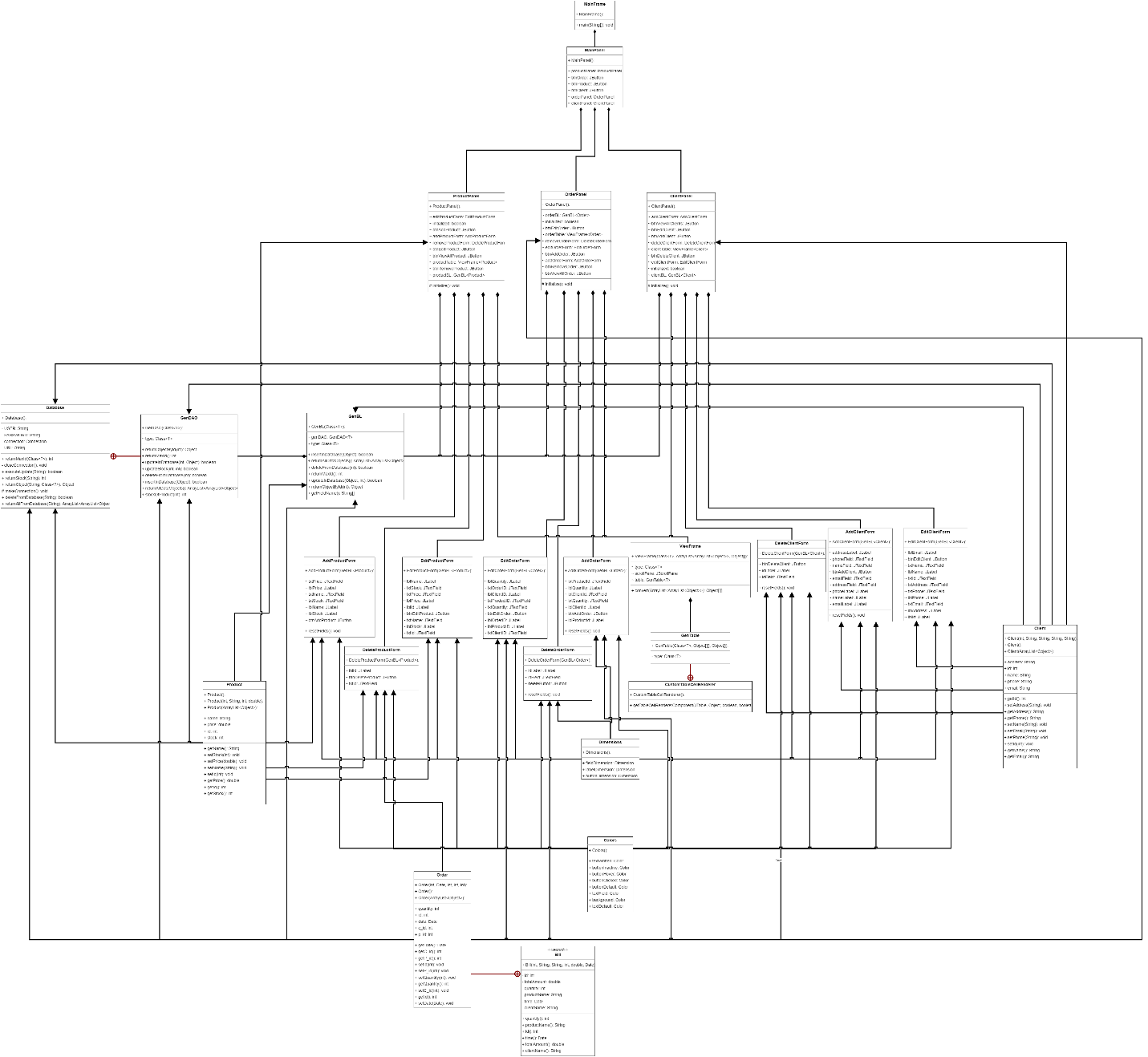
## OOP Design of the Application:

The application is designed using Object-Oriented Programming (OOP) principles, which include encapsulation, inheritance, and polymorphism. The application is divided into different classes, each focusing on a specific responsibility, making it modular and easy to maintain.



## UML Package and Class Diagrams:





The application consists of five packages: BusinessLogic, DataAccess, Model, Presentation and Util.

BusinessLogic:

* GenBL

DataAccess:

* GenDAO

Model:

* Order
* Product
* Client

Presentation:

* MainPanel
* ClientPanel
* ProductPanel
* OrderPanel
* AddProductForm
* AddClientForm
* AddOrderForm
* DeleteProductForm
* DeleteClientForm
* DeleteOrderForm
* EditProductForm
* EditClientForm
* EditOrderForm
* GenTable
* MainFrame

Util:

* Colors
* Dimensions

# Implementation

In this section I will present some details about the implementation of each class

1. Client Class

The Client class represents a client in the Orders Management application. It has the following attributes:

* id (int): The unique identifier of the client.
* name (String): The name of the client.
* address (String): The address of the client.
* email (String): The email address of the client.
* phone (String): The phone number of the client.

The class provides constructors for creating a Client object:

* Client(int id, String name, String address, String email, String phone): This constructor initializes a Client object with the provided values for the attributes.
* Client(): This is a default constructor that creates an empty Client object.
* The Client class also provides a constructor that takes an ArrayList<Object> as a parameter:
* Client(ArrayList<Object> objects): This constructor allows initializing a Client object from an ArrayList containing the attribute values in the specified order (id, name, address, email, phone).

The class includes getter and setter methods for accessing and modifying the attribute values:

* getId(): Returns the id of the client.
* setId(int id): Sets the id of the client.
* getName(): Returns the name of the client.
* setName(String name): Sets the name of the client.
* getAddress(): Returns the address of the client.
* setAddress(String address): Sets the address of the client.
* getEmail(): Returns the email address of the client.
* setEmail(String email): Sets the email address of the client.
* getPhone(): Returns the phone number of the client.
* setPhone(String phone): Sets the phone number of the client.

1. Order Class

The Order class represents an order in the Orders Management system. It stores information about the order, including its ID, date, client ID, product ID, and quantity.

The class provides constructors for creating an Order object:

* Order(int id, Date date, int c\_id, int p\_id, int quantity): This constructor initializes an Order object with the specified attributes.
* Order(): This is a default constructor that creates an empty Order object.
* Order(ArrayList<Object> objects): This constructor allows initializing an Order object from an ArrayList containing the attribute values in the specified order (id, date, c\_id, p\_id, quantity).

The class includes getter and setter methods for accessing and modifying the attribute values:

* getId(): Returns the ID of the order.
* setId(int id): Sets the ID of the order.
* getDate(): Returns the date of the order.
* setDate(Date date): Sets the date of the order.
* getC\_id(): Returns the ID of the client associated with the order.
* setC\_id(int c\_id): Sets the ID of the client associated with the order.
* getP\_id(): Returns the ID of the product associated with the order.
* setP\_id(int p\_id): Sets the ID of the product associated with the order.
* getQuantity(): Returns the quantity of the product in the order.
* setQuantity(int quantity): Sets the quantity of the product in the order.

1. Product Class

The Product class represents a product in the Products Management system. It encapsulates information about a product, including its ID, name, stock quantity, and price.

Attributes:

* id (int): The unique identifier of the product.
* name (String): The name of the product.
* stock (int): The current stock quantity of the product.
* price (double): The price of the product.

Constructors:

* Product(int id, String name, int stock, double price): This constructor initializes a new Product object with the specified attributes: ID, name, stock quantity, and price.
* Product(): This is a default constructor for the Product class that creates an empty Product object.
* Product(ArrayList<Object> values): This constructor allows initializing a Product object using an ArrayList of objects. The ArrayList should contain the ID, name, stock quantity, and price in the same order. It handles the conversion and assignment of values from the ArrayList to the respective attributes of the Product object.

Methods:

Getters and Setters: The class provides getter and setter methods for each attribute to retrieve and modify their values.

* getId(): Retrieves the ID of the product.
* setId(int id): Sets the ID of the product.
* getName(): Retrieves the name of the product.
* setName(String name): Sets the name of the product.
* getStock(): Retrieves the stock quantity of the product.
* setStock(int stock): Sets the stock quantity of the product.
* getPrice(): Retrieves the price of the product.
* setPrice(double price): Sets the price of the product.

1. GenDAO Class

The GenDAO class is a generic Data Access Object (DAO) class that provides generic database operations for different types of objects. It is parameterized with the class type T, which represents the type of the object.

The class has a nested Database class that encapsulates the database connection and provides various utility methods for executing database queries and retrieving data.

The GenDAO class has the following attributes:

* type (Class<T>): The class type of the object.

The class provides the following constructors:

* GenDAO(Class<T> type): Constructs a GenDAO object for the specified class type.

The GenDAO class includes the following methods:

* insertInDatabase(Object object): Inserts the specified object into the database.
* returnObjectById(int id): Retrieves an object from the database based on the specified ID.
* returnMaxId(): Returns the maximum ID from the database for the specified object type.
* deleteFromDatabase(int id): Deletes an object from the database based on the specified ID.
* updateInDatabase(int id, Object object): Updates an object in the database based on the specified ID.
* returnAllDataObjects(): Retrieves all data objects from the database.
* stockOfProduct(int id): Returns the stock quantity of a product with the specified ID.
* updateStock(int pId, int quantity): Updates the stock quantity of a product based on the specified ID and quantity.

The GenDAO class uses the Database class for executing SQL queries and interacting with the database. It includes methods such as executeUpdate, returnObject, returnMaxId, deleteFromDatabase, returnAllFromDatabase, and returnStock to perform database operations.

The GenDAO class serves as a generic data access layer for interacting with the database and provides common functionality for CRUD (Create, Read, Update, Delete) operations on objects of different types.

1. GenBL Class

The GenBL class is a generic business logic class that provides generic methods for interacting with the database. It is responsible for performing common CRUD (Create, Read, Update, Delete) operations on objects of a specific type.

The class has the following attributes:

* type (Class<T>): Represents the type of objects that the class will handle.
* genDAO (GenDAO<T>): An instance of the GenDAO class for performing data access operations.

The class provides the following constructor:

* GenBL(Class<T> type): Initializes the GenBL object with the specified type. It also creates an instance of the GenDAO class for the given type.

The GenBL class includes the following methods:

* getFieldName(): Retrieves the names of the fields of the specified type using reflection. It returns an array of strings representing the field names.
* returnAllDataObjects(): Retrieves all data objects of the specified type from the database using the genDAO object. It returns an ArrayList of ArrayList<Object> representing the retrieved data objects.
* insertInDatabase(Object object): Inserts an object into the database. For objects of type Order, it performs additional checks before inserting the order. It checks if the stock of the product is sufficient and updates the stock accordingly. Returns true if the object is successfully inserted, otherwise returns false.
* updateInDatabase(Object object, int id): Updates an object in the database based on the provided object and its identifier (id). It returns true if the object is successfully updated, otherwise returns false.
* deleteFromDatabase(int id): Deletes an object from the database based on its identifier (id). It returns true if the object is successfully deleted, otherwise returns false.
* returnMaxId(): Retrieves the maximum identifier (id) from the database for the specified type using the genDAO object. It returns an integer representing the maximum identifier.
* returnObjectById(int id): Retrieves an object from the database based on its identifier (id) using the genDAO object. It returns the retrieved object.

The GenBL class is designed to provide generic functionality for performing common database operations. It utilizes the GenDAO class for handling the data access layer and provides methods to interact with the database using the specified type.

1. GUI Classes

The GUI classes in the application provide the user interface for interacting with the system. They are responsible for displaying information, capturing user input, and triggering appropriate actions. The following GUI classes are available:

* 1. MainPanel

The MainPanel class represents the main panel of the application. It serves as the entry point for accessing different functionalities of the system, such as managing clients, products, and orders. The main panel provides navigation options to access the respective panels.

* 1. ClientPanel

The ClientPanel class displays a list of clients in the system. It allows users to view, add, edit, and delete client records. The panel communicates with the backend logic to perform the corresponding operations on client data.

* 1. ProductPanel

The ProductPanel class presents a list of products in the system. It enables users to view, add, edit, and delete product records. The panel interacts with the backend logic to perform the necessary operations on product data.

* 1. OrderPanel

The OrderPanel class shows a list of orders in the system. It provides functionalities for viewing, adding, editing, and deleting order records. The panel communicates with the backend logic to manage order data.

* 1. AddProductForm

The AddProductForm class extends JFrame and provides a form for adding a new product to the system. It includes labels and text fields to capture product information and a button to initiate the addition process. The class interacts with the business logic layer to insert the product into the database.

* 1. DeleteProductForm

The DeleteProductForm class extends JFrame and offers a form for deleting a product from the system. It includes a dropdown or a list of available products and a button to initiate the deletion process. The class interacts with the business logic layer to remove the selected product from the database.

* 1. EditProductForm

The EditProductForm class extends JFrame and provides a form for editing an existing product in the system. It displays the current product information and allows users to modify the fields. The class interacts with the business logic layer to update the product in the database.

* 1. AddOrderForm

The AddOrderForm class extends JFrame and provides a form for adding a new order to the system. It includes labels and text fields to capture order information and a button to initiate the addition process. The class interacts with the business logic layer to insert the order into the database.

* 1. DeleteOrderForm

The DeleteOrderForm class extends JFrame and offers a form for deleting an order from the system. It includes a dropdown or a list of available orders and a button to initiate the deletion process. The class interacts with the business logic layer to remove the selected order from the database.

* 1. EditOrderForm

The EditOrderForm class extends JFrame and provides a form for editing an existing order in the system. It displays the current order information and allows users to modify the fields. The class interacts with the business logic layer to update the order in the database.

* 1. AddClientForm

The AddClientForm class extends JFrame and provides a form for adding a new client to the system. It includes labels and text fields to capture client information and a button to initiate the addition process. The class interacts with the business logic layer to insert the client into the database.

* 1. DeleteClientForm

The DeleteClientForm class extends JFrame and offers a form for deleting a client from the system. It includes a dropdown or a list of available clients and a button to initiate the deletion process. The class interacts with the business logic layer to remove the selected client from the database.

* 1. EditClientForm

The EditClientForm class extends JFrame and provides a form for editing an existing client in the system. It displays the current client information and allows users to modify the fields. The class interacts with the business logic layer to update the client in the database.

* 1. GenTable

The GenTable class represents a generic table component that can be used to display data in a tabular format.

* 1. MainFrame

The MainFrame class extends JFrame and serves as the main frame of the application. It contains the main panels and manages the overall layout and navigation within the application. The class initializes the necessary components and provides a graphical user interface for users to interact with the system.

# Results

Based on the manual testing done on the software, I did not encounter problems. I let the app still print in console the queries for verification purpose

# Conclusions

The application works how I planned it. During the making of the project I leaned how to use reflection techniques and how to create generic classes that can replace a lot of classes. I also learned how to use the “new” data structure, the JTable. I can say that this assignment was the toughest out of the three from this year. It required a lot of debugging, attention and documentation to learn in order to understand the reflection techniques.

# Bibliography

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

<https://www.geeksforgeeks.org/reflection-in-java/>

<https://www.oracle.com/technical-resources/articles/java/javareflection.html>

<https://docs.oracle.com/javase/tutorial/uiswing/components/table.html>

<https://stackoverflow.com/questions/3875607/change-the-background-color-of-a-row-in-a-jtable>