DOCUMENTATION

ASSIGNMENT *3*

STUDENT NAME: Tcaci Liviu
GROUP: 30221

CONTENTS

1. Assignment Objective

Main Objective

The main objective of this assignment is to develop a comprehensive Orders Management application for processing client orders for a warehouse, adhering to the principles of layered architecture.

Sub-Objective	*Description*	*Addressed in Section*
1. Define the data models	*Create the data models for User, Product, and Order, representing the structure of the data in the application.*	*Section 4*
2. Implement the data access layer	*Develop Data Access Object (DAO) classes for each model to handle CRUD (Create, Read, Update, Delete) operations with the database.*	*Section 4*
3. Develop the business logic	*Create classes that encapsulate the core functionality of the application, such as order processing and business rules enforcement.*	*Section 4*
4. Design the GUI	*Develop a graphical user interface using Java Swing for managing users, products, and orders, including functionalities for adding, editing, deleting, and viewing data.*	*Section 4*
5. Use reflection for dynamic table creation	*Implement reflection techniques to dynamically generate table headers and populate rows based on object properties, enhancing the flexibility and reusability of the code.*	*Section 4*
6. Test the application	*Perform unit testing using JUnit and functional testing to verify that all components of the application work as expected. Document the results and ensure the application meets the specified requirements.*	*Section 5*

- ***Detailed Description of Sub-objectives***

Define the data models:

Objective: To accurately represent the data structures used in the application.
Description: Create Java classes for User, Product, and Order, each containing appropriate fields, constructors, getters, and setters.
Section: Implementation
Implement the data access layer:

Objective: To enable communication between the application and the relational database.
Description: Develop DAO classes (UserDAO, ProductDAO, OrderDAO) to handle CRUD operations. These classes will use JDBC to interact with the database.
Section: Implementation
Develop the business logic:

Objective: To encapsulate the core functionality and rules of the application.
Description: Create business logic classes that manage the creation of orders, validation of inputs, and application of business rules such as stock management and order processing.
Section: Implementation
Design the GUI:

Objective: To provide an intuitive and user-friendly interface for interacting with the application.
Description: Use Java Swing to develop a GUI with panels for managing users, products, and orders. The GUI will include forms for input, buttons for actions, and tables for displaying data.
Section: Implementation
Use reflection for dynamic table creation:

Objective: To dynamically generate table headers and rows based on the properties of objects.
Description: Implement a utility class that uses reflection to create tables from lists of objects, extracting field names for headers and field values for rows.
Section: Implementation
Test the application:

Objective: To ensure the application functions correctly and meets all specified requirements.
Description: Conduct unit tests for DAO methods using JUnit to verify CRUD operations. Perform functional testing to ensure the GUI interacts correctly with the underlying logic and database.
Section: Results

2. Problem Analysis, Modeling, Scenarios, Use Cases

Functional Requirements

The functional requirements of the Orders Management application are as follows:

1. *User Management:*

Add new users to the system.
Edit existing user details.
Delete users from the system.
View all users in a tabular format.

2. *Product Management:*

Add new products to the inventory.
Edit existing product details.
Delete products from the inventory.
View all products in a tabular format.

3. *Order Management:*

Create new orders by selecting a product and a user.
Ensure the quantity of the product is sufficient for the order.
Update the product stock after an order is created.
Display an under-stock message if there are not enough products.
View all orders in a tabular format.

4. *Dynamic Table Generation:*

Use reflection to dynamically generate the headers and rows of tables based on object properties.

Use Cases

Use Case 1: Add User

- *Actors: Admin/User*
- *Precondition: Admin/User is authenticated.*
- *Flow of Events:*

Admin/User selects the "Add User" option.
Admin/User fills in the user details (username, email, password, address).
Admin/User submits the form.
System saves the new user in the database.
System confirms the successful addition of the user.

- *Postcondition: The new user is added to the system.*

Use Case 2: Edit User

- *Actors: Admin/User*
- *Precondition: Admin/User is authenticated.*
- *Flow of Events:*

Admin/User selects an existing user to edit.
Admin/User updates the user details.
Admin/User submits the form.
System updates the user details in the database.
System confirms the successful update of the user.

- *Postcondition: The user's details are updated in the system.*

Use Case 3: Delete User

- *Actors: Admin/User*
- *Precondition: Admin/User is authenticated.*

- *Flow of Events:*

Admin/User selects an existing user to delete.
Admin/User confirms the deletion.
System deletes the user from the database.
System confirms the successful deletion of the user.

- *Postcondition: The user is deleted from the system.*

Use Case 4: View All Users

- *Actors: Admin/User*
- *Precondition: Admin/User is authenticated.*
- *Flow of Events:*

Admin/User selects the "View Users" option.
System fetches all users from the database.
System displays users in a JTable.

- *Postcondition: All users are displayed in a table.*

Use Case 5: Add Product

- *Actors: Admin/User*
- *Precondition: Admin/User is authenticated.*
- *Flow of Events:*

Admin/User selects the "Add Product" option.
Admin/User fills in the product details (quantity, name, price).
Admin/User submits the form.
System saves the new product in the database.
System confirms the successful addition of the product.

- *Postcondition: The new product is added to the system.*

Use Case 6: Edit Product

- *Actors: Admin/User*
- *Precondition: Admin/User is authenticated.*
- *Flow of Events:*

Admin/User selects an existing product to edit.
Admin/User updates the product details.
Admin/User submits the form.
System updates the product details in the database.
System confirms the successful update of the product.

- *Postcondition: The product's details are updated in the system.*

Use Case 7: Delete Product

- *Actors: Admin/User*
- *Precondition: Admin/User is authenticated.*
- *Flow of Events:*

Admin/User selects an existing product to delete.
Admin/User confirms the deletion.
System deletes the product from the database.
System confirms the successful deletion of the product.

- *Postcondition: The product is deleted from the system.*

Use Case 8: View All Products

- *Actors: Admin/User*
- *Precondition: Admin/User is authenticated.*
- *Flow of Events:*

Admin/User selects the "View Products" option.
System fetches all products from the database.
System displays products in a JTable.

- *Postcondition: All products are displayed in a table.*

Use Case 9: Create Order

- *Actors: Admin/User*
- *Precondition: Admin/User is authenticated.*
- *Flow of Events:*

Admin/User selects the "Create Order" option.
Admin/User selects a product and a user.
Admin/User inputs the order details (quantity).
System checks if there is enough stock.
If sufficient stock, system saves the order and updates product stock.
If insufficient stock, system displays an under-stock message.
System confirms the successful creation of the order.

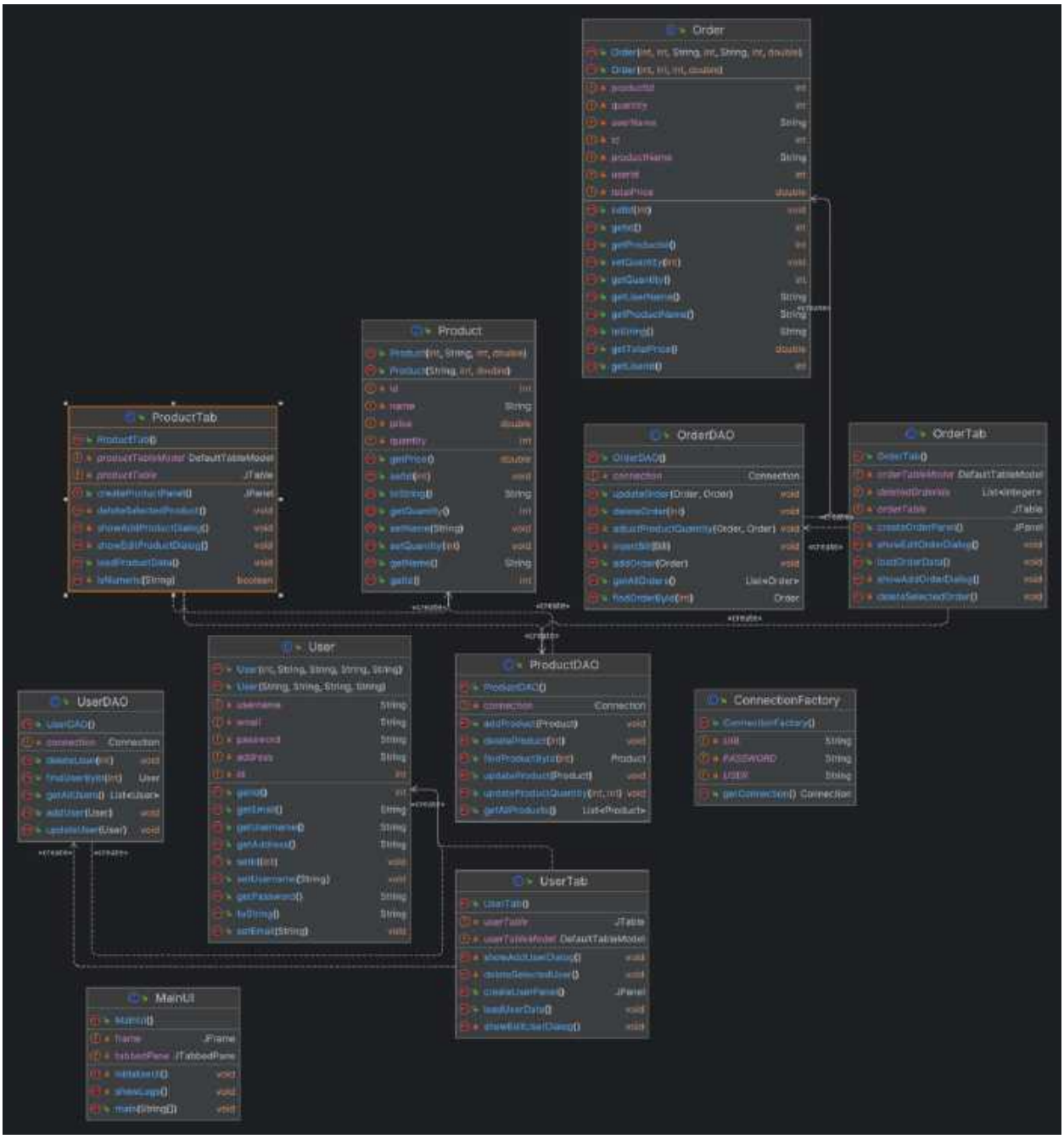
- *Postcondition: The order is created and product stock is updated.*

3. Design

OOP Design

The Orders Management application follows the principles of Object-Oriented Programming (OOP) and is structured into the following layers:

Model Layer: Contains the data models representing the entities in the application.
Data Access Layer: Contains the Data Access Object (DAO) classes responsible for interacting with the database.
Business Logic Layer: Contains the core functionality and business rules of the application.
Presentation Layer: Contains the graphical user interface components.



4. Implementation

In this section, we describe the implementation details of each class and component of the Orders Management application without including actual code. This includes the functionality, key methods, and how they interact with each other.

User Class
- *Description : Represents a user in the system.*
- *Fields :*
  - *username : The user's unique identifier.*
  - *email : The user's email address.*
  - *password : The user's password.*
  - *address : The user's address.*
- *Key Methods :*
  - *Constructor: Initializes a new User object with the given details.*
  - *Getters and Setters: Methods to retrieve and update the field values.*

Product Class
- *Description : Represents a product in the inventory.*
- *Fields :*
  - *quantity : The available quantity of the product.*
  - *name : The name of the product.*
  - *price : The price of the product.*
- *Key Methods :*
  - *Constructor: Initializes a new Product object with the given details.*
  - *Getters and Setters: Methods to retrieve and update the field values.*

Order Class
- *Description : Represents an order placed by a user.*
- *Fields :*
  - *quantity : The quantity of the product ordered.*
  - *totalPrice : The total price of the order.*
  - *user : The user who placed the order.*
  - *product : The product being ordered.*
- *Key Methods :*
  - *Constructor: Initializes a new Order object with the given details.*
  - *Getters and Setters: Methods to retrieve and update the field values.*

UserDAO Class
- *Description : Handles database operations for User entities.*
- *Key Methods :*
  - *addUser(User user) : Adds a new user to the database.*
  - *getUser(String username) : Retrieves a user by their username.*
  - *updateUser(User user) : Updates an existing user's details.*
  - *deleteUser(String username) : Deletes a user from the database.*
  - *getAllUsers() : Retrieves all users from the database.*

ProductDAO Class
- *Description : Handles database operations for Product entities.*
- *Key Methods :*
  - *addProduct(Product product) : Adds a new product to the database.*
  - *getProduct(String name) : Retrieves a product by its name.*
  - *updateProduct(Product product) : Updates an existing product's details.*
  - *deleteProduct(String name) : Deletes a product from the database.*
  - *getAllProducts() : Retrieves all products from the database.*

OrderDAO Class
- *Description : Handles database operations for Order entities.*
- *Key Methods :*
  - *addOrder(Order order) : Adds a new order to the database and updates product stock.*

TableUtils Class
- *Description : Utility class to create tables dynamically using reflection.*
- *Key Methods :*
  - *createTableFromList(List<T> dataList) : Generates a JTable from a list of objects by extracting field names for headers and field values for rows.*

Graphical User Interface (GUI)

- *MainUI Class*
  - *Description : The main window of the application.*
  - *Components :*
    - *Buttons for adding, editing, deleting, and viewing users and products.*
    - *A JTable for displaying data dynamically.*
  - *Functionality : Initializes the main interface and sets up action listeners for user interactions.*

- *UserTab Class*
  - *Description : Manages user-related operations.*
  - *Components :*
    - *Forms for adding and editing users.*
    - *A JTable for displaying user data.*
  - *Functionality : Handles user input and displays user data.*

- *ProductTab Class*
  - *Description : Manages product-related operations.*
  - *Components :*
    - *Forms for adding and editing products.*
    - *A JTable for displaying product data.*
  - *Functionality : Handles product input and displays product data.*

- *OrderTab Class*
  - *Description : Manages order-related operations.*
  - *Components :*
    - *Forms for creating orders.*
    - *A JTable for displaying order data.*
  - *Functionality : Handles order input and displays order data.*

5. Results

Testing Scenarios

- *Unit Testing :*
  - *Objective : To verify the correctness of individual methods in the DAO classes.*
  - *Approach : Use JUnit to write test cases for each CRUD operation in the DAO classes.*
  - *Results : All unit tests should pass, confirming that the DAO methods work correctly.*

- *Functional Testing :*
  - *Objective : To verify that the application works as a whole.*
  - *Approach : Perform manual testing of the GUI to ensure that all functionalities (adding, editing, deleting, and viewing users and products, creating orders) work as expected.*
  - *Results : The application should correctly handle user inputs, interact with the database, and display data in the GUI.*

6. Conclusions

What You Have Learned

- *Layered Architecture : How to structure a Java application using a layered architecture pattern, separating concerns into distinct layers (model, data access, business logic, and presentation).*
- *Java Reflection : How to use Java reflection to dynamically generate table headers and rows, making the application more flexible and reusable.*
- *Swing GUI Development : How to develop a user-friendly graphical user interface using Java Swing, handling user interactions and displaying data dynamically.*
- *Database Interaction : How to perform CRUD operations using JDBC, ensuring efficient and secure data access.*
- *Unit Testing : The importance of unit testing and how to write test cases to verify the correctness of individual methods.*

Future Developments

- *Advanced Features : Implement more advanced features like filtering and sorting of data in the tables, and exporting data to various formats (e.g., CSV, PDF).*
- *Improved GUI : Enhance the user interface for better usability and aesthetics, possibly using more advanced libraries or frameworks.*
- *Error Handling and Validation : Add more robust error handling and input validation to improve the reliability and security of the application.*
- *Scalability : Refactor the application to handle larger datasets and more concurrent users, potentially integrating with more advanced database systems or using ORM frameworks.*

7. Bibliography

1. Oracle "The Java Tutorials."
2. Eckel, Bruce. *Thinking in Java (4th Edition)
3. JUnit Team"JUnit 5 User Guide."
4. How to Create a Test on IntelliJ (JUnit)
5. *"What are Java classes?"*
6. *Java Swing Tutorials*