DOCUMENTATION

ASSIGNMENT *ASSIGNMENT\_3*

STUDENT NAME: Săvianu Bogdan-Casian

GROUP: 30223

# CONTENTS

[1. Assignment Objective 3](#_Toc128043139)

[2. Problem Analysis, Modeling, Scenarios, Use Cases 3](#_Toc128043140)

[3. Design 3](#_Toc128043141)

[4. Implementation 3](#_Toc128043142)

[5. Results 3](#_Toc128043143)

[6. Conclusions 3](#_Toc128043144)

[7. Bibliography 3](#_Toc128043145)

# Assignment Objective

The main objective of this assignment was to create an application to help with the order management and inventory of a warehouse.

The making of this application was split into a few smaller goals: creating the database of the warehouse, the java backend which helps as an intermediate layer in interacting with the database, the Javadoc files for all of the classes where I explain the role of each class and describe the methods contained in the classes and a graphical user interface to make it easier to visualize the orders, products etc., and interact with them.

# Problem Analysis, Modeling, Scenarios, Use Cases

I modelled the problem as 3 tables in the database, and 3 corresponding classes in the Java application, namely Orders, Product, and Customer.

There are a few actions that can be done in the application: we can add, update and delete customers, products and orders. But when handling an order, things are a little more complex:

* When adding an order, we are asked if it is for a new customer or for an already existing one.
  + If we choose a new customer, we are prompted to enter their details and the customer is created, then we can proceed with the order;
  + If we want an existing customer, we can choose from a drop-down menu and then go on normally.
* When updating a customer/product/order, we click the desired one and then we are prompted to change all of the fields, but if we do not change a field, it stays unmodified.
* When deleting, like updating, we click on the object and then press ‘delete’. When deleting an order, the quantity of the product will be restored back to the warehouse with the help of a trigger in MySQL. When deleting a product, or a customer, their associated orders are deleted too.

# Design

A blurry image of a diagram

Description automatically generated

A diagram of a computer

Description automatically generated

# Implementation

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Model**

* **Customer** – represents the Customer table in the database with its fields, getters and setters
* **Orders** – represents the Orders table in the database
* **Product** – represents the Product table in the database

**DAO**

* **AbstractDAO –** provides the CRUD methods in a generic way so that all other classes that extends this class, ca utilize these methods as they are.
* **CustomerDAO –** provides the CRUD methods for customers
* **OrderDAO –** provides the CRUD methods for orders
* **ProductDAO –** provides the CRUD methods for products

These classes are meant to be a layer between the application and the database which provide access to the tuples in the database

**BLL**

* **CustomerBLL –** offers the business logic for handling customers: adding, updating, deleting customers
* **OrderBLL –** offer the business logic for handling orders
* **ProductBLL –** offers the business logic for handling products

**Controller**

* **CustomerController:**
  + *handleAddCustomer*: - adds the functionality to create a customer when the ‘Add’ button is pressed in the GUI
  + *handleUpdateCustomer* - adds the functionality to update a customer when the ‘Update’ button is pressed in the GUI
  + *handleDeleteCustomer* - adds the functionality to delete a customer when the ‘Delete’ button is pressed in the GUI
  + *initData* **–** initializes the table by calling the findAll method from CustomerBLL and listing all of the customers
* **OrderController:**
  + *handleAddOrder*: - adds the functionality to create an order when the ‘Add’ button is pressed in the GUI
  + *handleUpdateOrder* - adds the functionality to update an order when the ‘Update’ button is pressed in the GUI
  + *handleDeleteOrder* - adds the functionality to delete an order when the ‘Delete’ button is pressed in the GUI
  + *initData* **–** initializes the table by calling the findAll method from OrderBLL and listing all of the orders
* **ProductController:**
  + *handleAddProduct*: - adds the functionality to create a product when the ‘Add’ button is pressed in the GUI
  + *handleUpdateProduct* - adds the functionality to update a product when the ‘Update’ button is pressed in the GUI
  + *handleDeleteProduct* - adds the functionality to delete a product when the ‘Delete’ button is pressed in the GUI
  + *initData* **–** initializes the table by calling the findAll method from ProductBLL and listing all of the products

**SinglePointAccess**

* **RepoSinglePointAccess:** provides global access to static instances of CustomerBLL, ProductBLL, OrderBLL so as to respect the Singleton design pattern.

# Results

To make sure that the CRUD methods work, I tested them in the Main class where I created some dummy objects that I added to the database, then I updated them and, finally, I tried deleting them.

After I made sure they worked well, I was able to move forward and implement the functionality of these methods into the buttons in the graphical user interface.

# Conclusions

From this assignment I learned what reflection means and the benefits it brings to the table, namely, in this case, accessing the fields of the model classes in a generic way where, if I modify the model, reflection helps me not have to modify the CRUD methods too much.

Also, this was my first time using Javadoc files and it felt like being one step closer to working in the industry.

# Bibliography

<https://dsrl.eu/courses/pt/>

### **[Java Reflection Explained - bɘniɒlqxƎ noiɟɔɘlʇɘЯ ɒvɒᒐ](https://www.youtube.com/watch?v=bhhMJSKNCQY)**

[](https://www.youtube.com/watch?v=bhhMJSKNCQY)

[YouTube · Coding with John](https://www.youtube.com/watch?v=bhhMJSKNCQY)

[166.2K+ views · 1 year ago](https://www.youtube.com/watch?v=bhhMJSKNCQY)

### **[Using Java Reflection](https://www.oracle.com/technical-resources/articles/java/javareflection.html)**

[](https://www.oracle.com/technical-resources/articles/java/javareflection.html)

[Oracle](https://www.oracle.com/technical-resources/articles/java/javareflection.html)

[https://www.oracle.com › Java › Technical Details](https://www.oracle.com/technical-resources/articles/java/javareflection.html)

### **[Guide to Java Reflection](https://www.baeldung.com/java-reflection)**

[](https://www.baeldung.com/java-reflection)

[Baeldung](https://www.baeldung.com/java-reflection)

[https://www.baeldung.com · Jan 16, 2024](https://www.baeldung.com/java-reflection)

### **[Introduction to JavaDoc](https://www.baeldung.com/javadoc)**

[](https://www.baeldung.com/javadoc)

[Baeldung](https://www.baeldung.com/javadoc)

[https://www.baeldung.com › javadoc](https://www.baeldung.com/javadoc)