

# DOCUMENTATION

## ASSIGNMENT 3

STUDENT NAME: RUNCAN PAUL-MARIUS

GROUP: e\_30422

# CONTENTS

1.	Assignment Objective .....	3
2.	Problem Analysis, Modeling, Scenarios, Use Cases.....	4
3.	Design .....	7
4.	Implementation .....	14
5.	Results.....	20
6.	Conclusions.....	21
7.	Bibliography .....	21

# 1. Assignment Objective

## *Main Objective*

- Design and implement an application for managing the client orders for a warehouse in which orders/clients/products are stored in a database

## *Secondary Objectives:*

- Analyze the problem and identify the requirements
- Design the orders management application
- Implement the orders management application
- Test the orders application

## 2. Problem Analysis, Modeling, Scenarios, Use Cases

### **Functional requirements:**

The managing application should:

- allow the user to insert a new customer/product.
- allow the user to select a customer, select a product, enter a quantity of the product and create a order with the given specifications
- print an error message if the quantity isn't available
- allow the user to create a Bill for each order inserted.
- Save the created data in a database

### **Non-Functional requirements:**

The managing application should:

- be intuitive, it must be clear for the user where the data should be inserted and it should be clear what each table shown in the GUI contains
- be reliable, failures should appear only if the data types aren't respected.
- be maintainable, issues should be easily fixed because the code is well-structured and is based on simple algorithms.
- be secure, the application should be secure, having all fields private and inaccessible from the exterior.

## Use Cases:

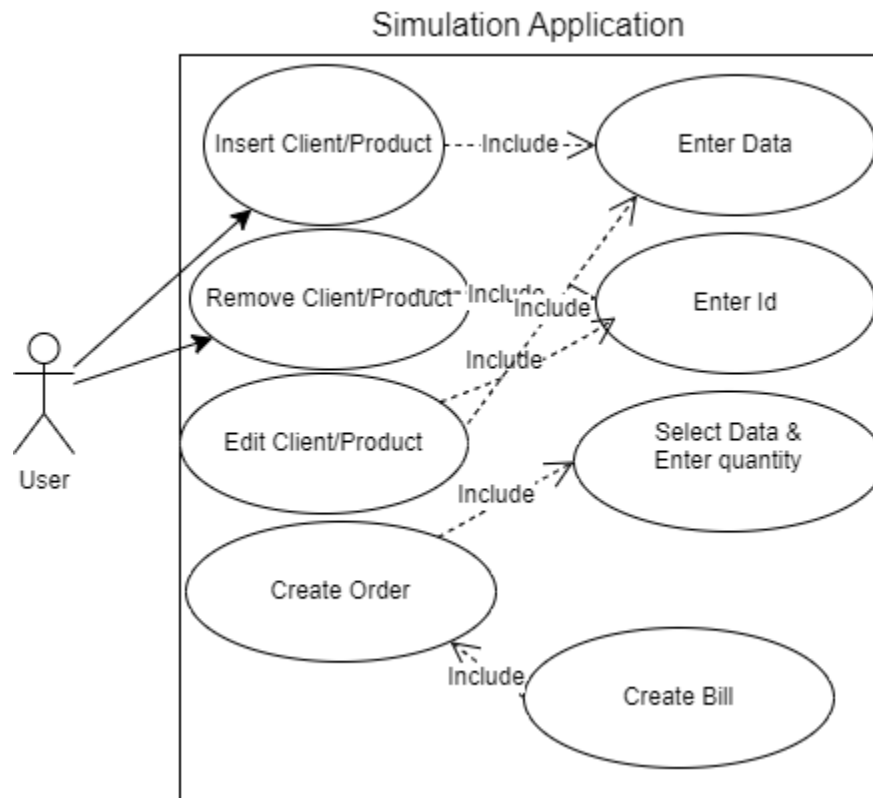


Figure 1. Use Case Diagram

**Primary actor:** User

**Main Success Scenario: Insert Customer/Product**

1. The user inserts correct data for inserting a new customer/product in the fields from the Graphical User Interface.
2. After pressing the “ADD” button, a new entry corresponding to the customer/product will be added to the database.

**Main Success Scenario: Delete Customer/Product**

1. The user inserts an existing id in the field from the Graphical User Interface then presses the delete button.
2. The entry corresponding to that id will be deleted from the database.

**Main Success Scenario: Edit Customer/Product:**

1. The user inserts an existing id in the field from the Graphical User Interface.

2. The user inserts the data that wants to be modified in the fields from the Graphical User Interface.
3. The user presses the edit button and the entry will be modified in the database.

**Main Success Scenario: Create Order:**

1. The user selects a name for the customer, a name for the product, and a quantity for the product.
2. The user presses the add product button and a new entry will be created in the database.

**Alternative Scenario:**

1. The user enters a unavailable quantity and presses the button.
2. A pop-up window appears with an error message.

## 3. Design

### Level 1: Overall System Design



**Figure 1. System Black Box**

The system consists of a different number of inputs depending on the operation that the user wants to be performed:

- Add Customer : name, mail;
- Add Product: name, quantity;
- Add Order: customer name, product name, quantity;
- Remove Customer/Product: id;
- Edit Customer: id, name, mail;
- Edit Product: id, name, quantity;

## Level 2: Design in sub-systems/packages

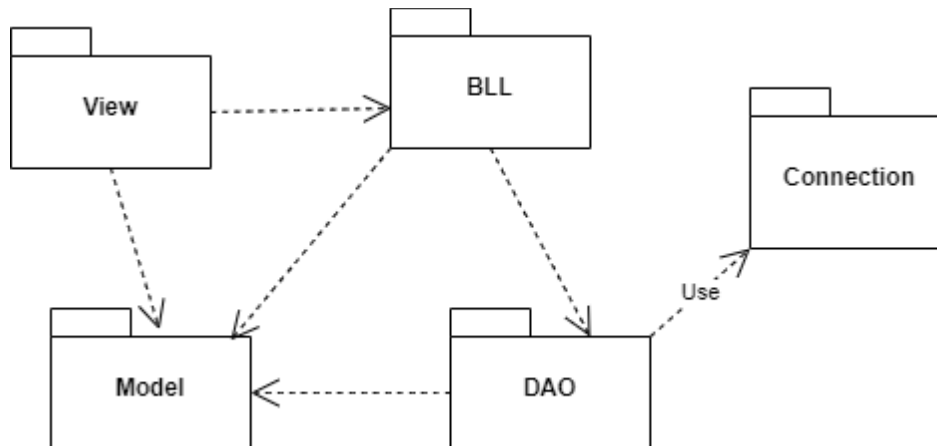


Figure 2. Package Diagram

**View:** contains the classes that implement the Graphical User Interface

**BLL:** contains the classes that handle the adding, removing, editing.

- CustomerBLL: handles the adding, removing editing of customers.
- ProductBLL: handles the adding, removing editing of products.
- OrderBLL: handles the adding of orders.
- BillBLL: handles the creation of bills.

**DAO:** contains the classes that handle the database access .

- AbstractDAO<T>: is the base class that is a generic class that handles the database access for all the operations.
- All other DAO classes extend the generic class and use the AbstractDAO's methods.

**Connection:** contains the Connection class which establishes a connection to the database and closes that connection, resultSet and statement.

**Model:** contains the classes that model the Customer, Product, Order and Bill.



### Level 3: Division into classes/Class Diagram.

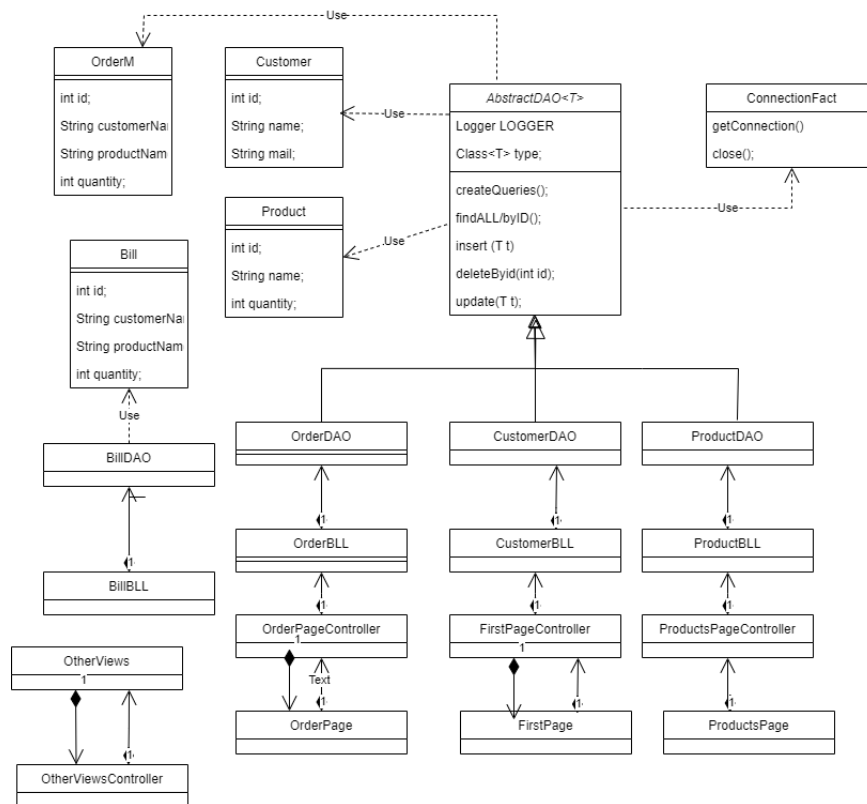


Figure 3. Class Diagram

### Used Data Structures:

Lists were used for storing the entries from the database tables when we needed to use them.

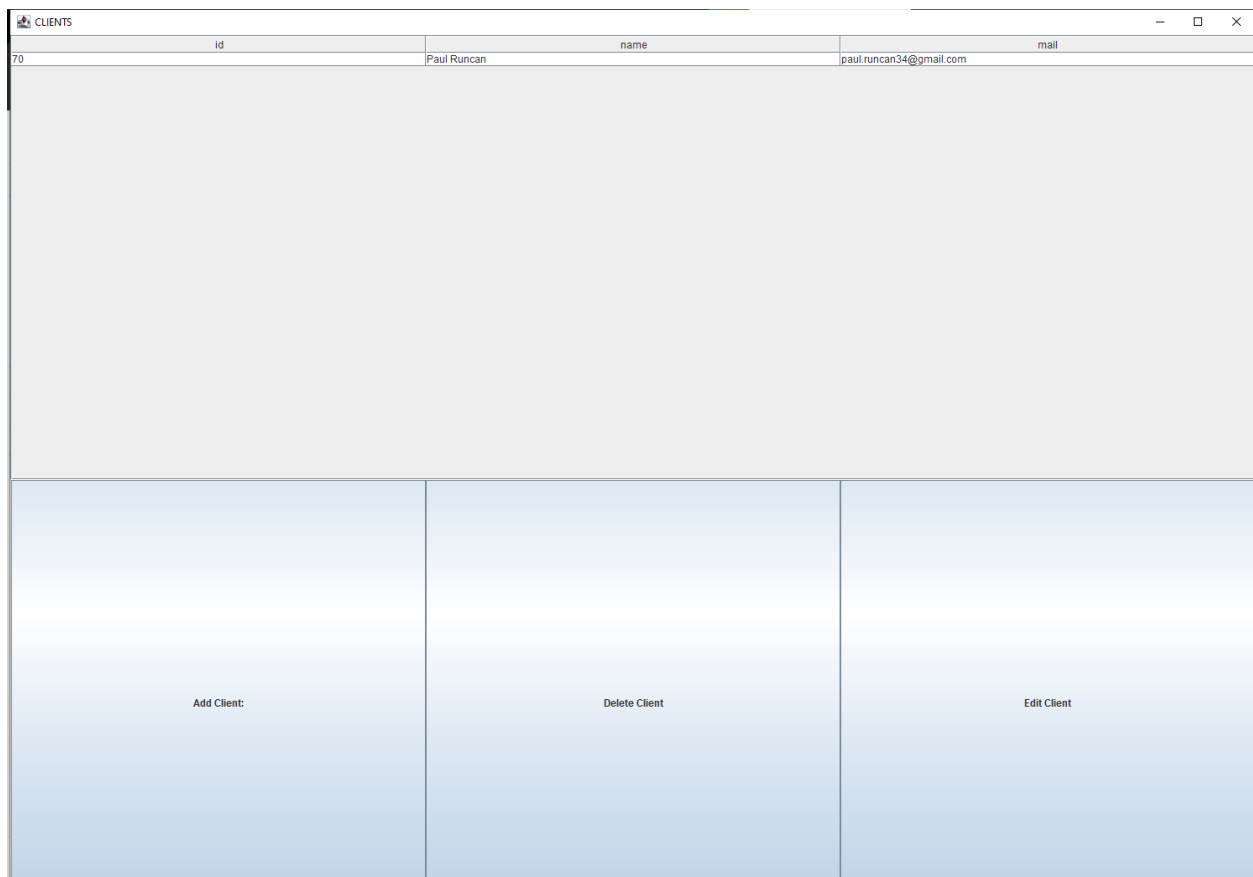
ResultSets were used to extract the entries from the database tables.

## GUI Design:


### Main Page:




### Clients/Product Page:




### Add/Edit/Delete Pages:

 ADD — □ ×

name
mail
ADD

 R... — □ ×

id
REMOVE

 EDIT — □ ×

id	name
mail	EDIT

Orders Page:

ORDERS

id	customerName	productName	quantity
1	adasda	dasdasfa	123
2	georgica	gole	12
3	dfasdfasdf	sampon	12
4	georgica	gole	10
5	adsadas	gole	10
6	1313123sadsasd	ananas	250
7	fasfasfaf	ananas	250
8	2131231	ananas	250
9	2131231	ananas	250
10	2131231	ananas	250
11	1313123sadsasd	sampon	20
12	fasdfasd	ananas	24
13	asdasda	ananas	24
14	dadada	ananas	54
15	Paul Runcan	strawberry	25
16	Paul Runcan	strawberry	25

Add Order

Add Order Page:

DAAA

id	name	mail	quantity
70	Paul Runcan	paul.runcan34...	

id	name	quantity
9	ananas	1648
10	mere	2423
8	ampons	80
12	strawberry	0

ADD

BILLS

Bills Page:

BILLS

id	customerName	productName	quantity
1	andrei	apa	20
2	andrei	apa	20
3	fasdasa	ananas	24
4	asdasda	ananas	24
5	dadada	ananas	54
6	Paul Runcan	strawberry	25
7	Paul Runcan	strawberry	25

## 4.Implementation

### Classes:

#### View Package classes:

Contains the classes that create the view of the application:

- Table creation;

- Button creation;

Contains the classes that are controllers for the views:

- Action listeners. Give the behavior of the application when a button is pressed.

#### BLL Package classes:

**CustomerBLL:** contains an instance of CustomerDAO that helps with gathering data from the database.

Contains the methods for adding, deleting and editing a customer.

```
public Customer addCustomer( String customerName,String customerMail){
    Customer customer = customerDAO.insert(new
Customer(customerName,customerMail));
    return customer;
}

public void removeCustomerById(int id){
    customerDAO.deleteById(id);
}

public void editCustomer(int id, String name, String mail){
    customerDAO.update(new Customer(id,name,mail));
}
```

**ProductBLL:** contains an instance of ProductDAO that helps with gathering data from the database.

Contains the methods for adding, deleting and editing a customer.

```
public Product addProduct( String name, int quantity ) {
    Product product = productDAO.insert(new Product(name,quantity));
    return product;
}

public void removeProductById( int id ) {
    productDAO.deleteById(id);
}

public void editProduct( int id, String name, int quantity ) {
    productDAO.update(new Product(id,name,quantity));
}
```

**OrderBLL:** contains an instance of OrderDAO that helps with gathering data from the database.

Contains the method for adding an order to the database.

```
public OrderM addOrder( String customerName, String productName, int
neededQuantity ) {
    OrderM orderM = orderDAO.insert(new
OrderM(customerName,productName,neededQuantity));
    return orderM;
}
```

## Model Package:

**Bill: Field:** int id, String customerName, String productName, int quantity.

**Customer: Field:** int id, String name, String mail.

**Product: Field:** int id, String name, int quantity.

**Order: Field:** int id, String customerName, String productName, int quantity.

## DAO Package:

**AbstractDAO:** contains the methods for query creation.

```
private String createSelectQuery(String field) {
    StringBuilder sb = new StringBuilder();
    sb.append("SELECT ");
    sb.append(" * ");
    sb.append(" FROM ");
    sb.append(type.getSimpleName());
    sb.append(" WHERE " + field + " =?");
    return sb.toString();
}

public String createAddQuery() {
    StringBuilder sb = new StringBuilder();
    sb.append("INSERT INTO ");
    sb.append(type.getSimpleName());
    sb.append("(");
    for(Field field : type.getDeclaredFields())
        if(field.getName()!="id")
            sb.append(field.getName()).append(",");
    sb.deleteCharAt(sb.length()-1);
    sb.append(")");
    sb.append(" values(");
    sb.append("?, ".repeat(type.getDeclaredFields().length-1));
    sb.deleteCharAt(sb.length()-1);
    sb.append(")");
}
```

```

        System.out.println(sb.toString());
        return sb.toString();
        //return null;
    }

    public String createRemoveQuery(String field){
        StringBuilder sb =new StringBuilder();
        sb.append("DELETE FROM ");
        sb.append(type.getSimpleName());
        sb.append(" WHERE " + field + " =?");
        return sb.toString();
    }

    public String createUpdateQuery(String field){
        StringBuilder sb = new StringBuilder();
        sb.append("UPDATE ");
        sb.append(type.getSimpleName());
        sb.append(" set ");
        for(Field field1: type.getDeclaredFields())
            if(field1.getName() != "id")
                sb.append(field1.getName()).append("=?,");
        sb.deleteCharAt(sb.length()-1);
        sb.append(" WHERE " + field + "=?");
        return sb.toString();
    }

    private String createAllQuery() {
        StringBuilder sb = new StringBuilder();
        sb.append("SELECT ");
        sb.append("* ");
        sb.append("from ");
        sb.append(type.getSimpleName());
        //sb.append(" WHERE ")
        return sb.toString();
    }
}

```

Contains the methods for adding, editing, removing data from the database

```

public List<T> findAll() {
    Connection connection =null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    String query = createAllQuery();
    try {
        connection = ConnectionFact.getConnection();
        statement = connection.prepareStatement(query);
        resultSet = statement.executeQuery();

        return createObjects(resultSet);
    } catch (SQLException e) {
        LOGGER.log(Level.WARNING, type.getName() + "DAO:findById " +
e.getMessage());
    } finally {
        ConnectionFact.close(resultSet);
        ConnectionFact.close(statement);
        ConnectionFact.close(connection);
    }
}

```



```

        return null;
    }

    public T findById(int id) {
        Connection connection = null;
        PreparedStatement statement = null;
        ResultSet resultSet = null;
        String query = createSelectQuery("id");
        try {
            connection = ConnectionFact.getConnection();
            statement = connection.prepareStatement(query);
            statement.setInt(1, id);
            resultSet = statement.executeQuery();

            return createObjects(resultSet).get(0);
        } catch (SQLException e) {
            LOGGER.log(Level.WARNING, type.getName() + "DAO:findById " +
e.getMessage());
        } finally {
            ConnectionFact.close(resultSet);
            ConnectionFact.close(statement);
            ConnectionFact.close(connection);
        }
        return null;
    }

    public T insert(T t) {
        Connection connection = null;
        PreparedStatement statement = null;
        ResultSet resultSet = null;
        String query = createAddQuery();
        try {
            connection = ConnectionFact.getConnection();
            statement=connection.prepareStatement(query);
            int i=1;
            for( Field field : t.getClass().getDeclaredFields() )
            {
                field.setAccessible(true);
                if(field.getName()!="id"){
                    statement.setObject(i++,field.get(t));
                }
                System.out.println(field.getName());
                System.out.println(field.get(t));
            }
            statement.executeUpdate();
        } catch (Exception e){
            System.out.println("error");
            e.printStackTrace();
        }
        return t;
    }

    public void deleteById(int id){
        Connection connection = null;
        PreparedStatement statement = null;
        String query = createRemoveQuery("id");
        try {
            connection = ConnectionFact.getConnection();
            statement = connection.prepareStatement(query);

```

```

        statement.setInt(1, id);
        statement.executeUpdate();
    } catch (SQLException e) {
        LOGGER.log(Level.WARNING, type.getName() + "DAO:findById " +
e.getMessage());
    } finally {
        ConnectionFact.close(statement);
        ConnectionFact.close(connection);
    }
}

public T update(T t) {
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    String query = createUpdateQuery("id");
    try {
        connection = ConnectionFact.getConnection();
        statement=connection.prepareStatement(query);
        int i=1;
        for( Field field : t.getClass().getDeclaredFields() )
        {
            field.setAccessible(true);
            if(field.getName()!="id")
                statement.setObject(i++,field.get(t));
            System.out.println(field.getName());
            System.out.println(field.get(t));
        }
        for( Field field : t.getClass().getDeclaredFields() )
        {
            field.setAccessible(true);
            if(field.getName()=="id")
                statement.setObject(i++,field.get(t));
            System.out.println(field.getName());
            System.out.println(field.get(t));
        }
        statement.executeUpdate();
    } catch (Exception e){
        System.out.println("error");
        e.printStackTrace();
    }

    return t;
}

```

**BillDAO:** contains the methods for accessing data from Bill table.

```

public void insert( Bill bill ){
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    String query = inserString;
    try {
        connection = ConnectionFact.getConnection();
        statement=connection.prepareStatement(query);
        statement.setString(1, bill.customerName());
    }
}

```

```

        statement.setString(2, bill.productName());
        statement.setInt(3, bill.quantity());
        statement.executeUpdate();
    } catch (Exception e) {
        System.out.println("error");
        e.printStackTrace();
    }
}

public List<Bill> findAll() {
    Connection connection = null;
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    List<Bill> bills = new ArrayList<>();
    String query = selectAllString;
    try {
        connection = ConnectionFact.getConnection();
        statement = connection.prepareStatement(query);
        resultSet = statement.executeQuery();

        while (resultSet.next()) {
            int id = resultSet.getInt("id");
            String customerName = resultSet.getString("customerName");
            String productName = resultSet.getString("productName");
            int quantity = resultSet.getInt("quantity");
            System.out.println(customerName);
            System.out.println(productName);
            System.out.println(quantity);
            Bill bill = new Bill(id, customerName, productName, quantity);
            bills.add(bill);
        }
        return bills;
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        ConnectionFact.close(resultSet);
        ConnectionFact.close(statement);
        ConnectionFact.close(connection);
    }
    return null;
}

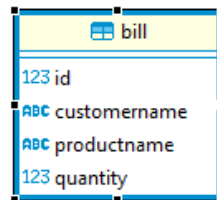
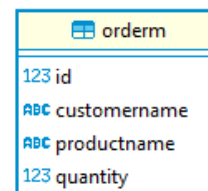
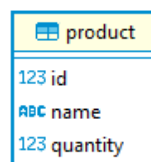
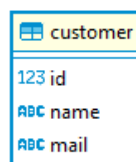
```

## 5.Results

Testing can be done through the GUI following the recommended steps.

The Results can be seen in the GUI and also in the Tables from the Database

properties ER Diagram





customer Enter a SQL expression to filter results (use Ctrl+Space)



	123 id	ABC name	ABC mail
1	70	Paul Runcan	paul.runcan34@gmail.com

orderm Enter a SQL expression to filter results (use Ctrl+Space)

	123 id	ABC customername	ABC productname	123 quantity
1	1	adasda	dasdasfa	123
2	2	georgica	gole	12

 **product** |  Enter a SQL expression to filter results (use Ctrl+).

	123 id ▼	ABC name ▼	123 quantity ▼
1	9	ananas	1,648
2	10	mere	2,423
3	8	ampons	80
4	12	strawberry	0

 **bill** |  Enter a SQL expression to filter results (use Ctrl+Space)

	123 id ▼	ABC customername ▼	ABC productname ▼	123 quantity ▼
1	1	andrei	apa	20
2	2	andrei	apa	20

## 6.Conclusions

In conclusion, the Order Manager has been implemented using Generic Classes, Reflection Technique and Records.

The Graphical User Interface was implemented using Java Swing.

Working on this project has helped me to develop my knowledge about reflection techniques, about working with the Layered Architecture and working with relational databases.

## 7.Bibliography

- [1] <https://dsrl.eu/courses/pt/> -> PT\_2024\_A3\_S1, PT\_2024\_A3\_S2, Java Swing, Lectures
- [2] <https://www.baeldung.com/java-jdbc>
- [3] <https://www.baeldung.com/javadoc>