OOSD Project Documentation

An Object-Oriented Application for Managing Customers, Products, and Invoices

Module Name: OOSD
Lecturer: Dr. Jason Barron

Course Code: CW_KCSOF_B
Name: Amelia Hamulewicz
Student ID: C00296605

Date of Submission: 10 April 2025

Table of Contents

Description

Overview

This project is a desktop app for managing products, customers, and invoices. It was built using Java Swing for the GUI and JDBC (Java Data Base Connection) to connect to a database. Users can add, edit, view, and delete products, customers, and invoices. The app works with a database that has at least three tables: Customer, Product, and Invoice.

Main Features

- **Product Management:**
  Users can add new products, update existing product details, delete products, and view a full list of all products.

- **Customer Management:**
  The application allows users to manage customer records, including adding, editing, and deleting customer information. Input validation ensures the data entered is correct and complete.

- **Invoice Creation:**
  Users can create new invoices, add or remove products from an invoice, and the system automatically updates stock levels based on item quantities.

- **Filtering and Searching:**
  Products and invoices can be filtered by category or customer to make it easier to find specific information.

- **Validation and Error Handling:**
  The system includes checks to ensure valid input, such as making sure prices and stock quantities are non-negative numbers. Errors are handled gracefully with clear messages.

- **Graphical User Interface (GUI):**
  The interface is built using Java Swing components such as JTable, JTextField, JComboBox, and JButton, offering a user-friendly and interactive experience.

Technologies Used

- **Java (JDK 8):**
  The core programming language used to develop the application, following object-oriented principles.
- **Java Swing:**
  Used to build the graphical user interface, including forms, tables, buttons, and input fields.
- **JDBC (Java Database Connectivity):**
  Enables the application to connect to and interact with a relational database through SQL queries.
- **SQL (MySQL):**
  Used for creating and managing the database tables, as well as executing queries for storing and retrieving data.

Requirements

Backend Requirements

- The system uses a **relational database** (e.g. MySQL or MariaDB).
- The database must contain at least **three tables**: Customer, Product, and Invoice.
- Tables are linked using **foreign keys**, and at least one **INNER JOIN** is used for queries (e.g. loading invoice details with customer info).
- A **JDBC driver** must be available to allow the Java app to connect to the database.

Frontend Requirements

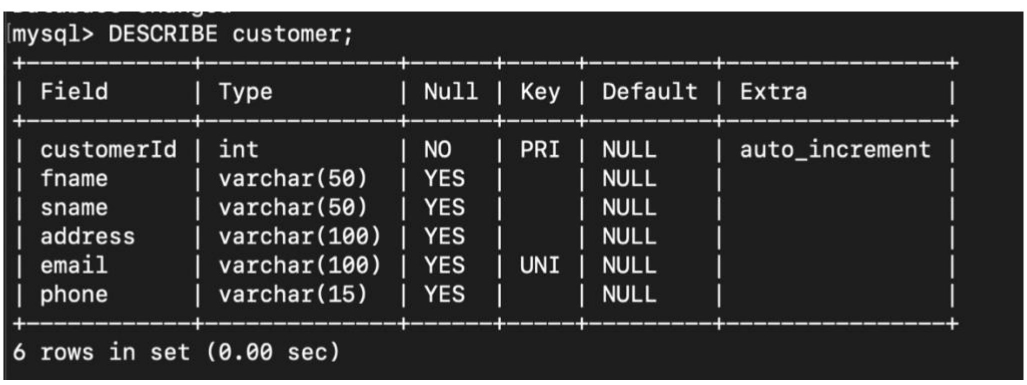
- The user interface is built with **Java Swing**, using components such as JButton, JTable, JComboBox, and JTextField.
- Users should be able to **create**, **view**, **update**, and **delete** products, customers, and invoices.
- Input fields must include **validation** to prevent invalid data (e.g. letters in number fields, empty fields, etc.).
- Error messages are displayed clearly when something goes wrong.

System & Setup

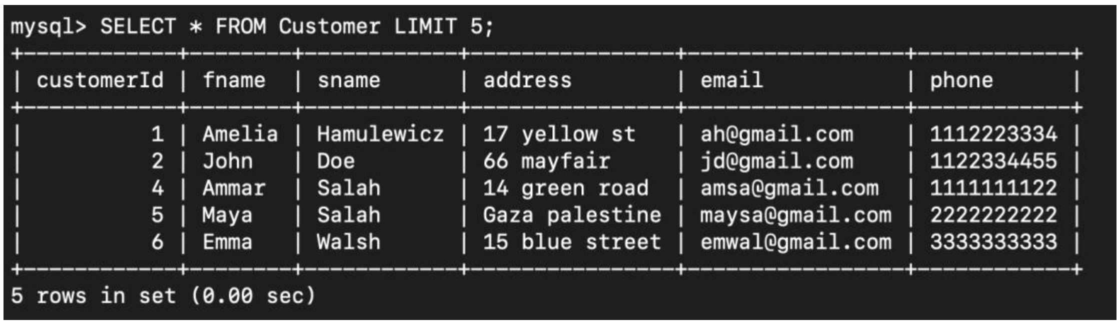
- **Java JDK 8 or higher** must be installed on the system.
- The system should support **Windows, macOS, or Linux**.
- Database connection details (host, username, password) are handled in the MyConnection class.

Screenshot Database Table (Structure and Data)

Customer Table

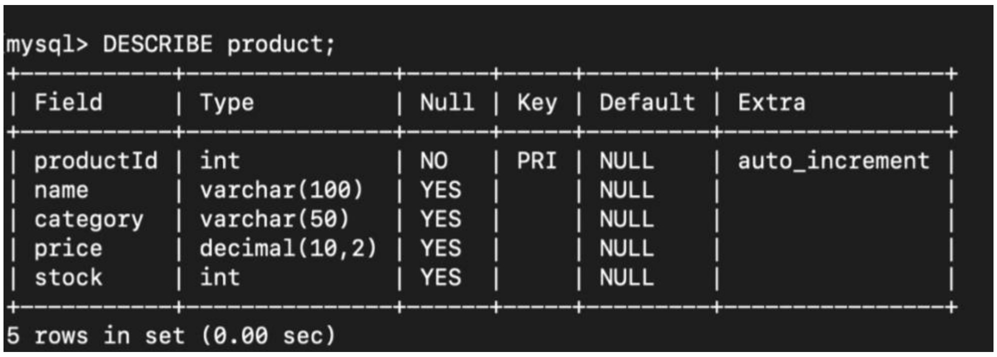


```
mysql> DESCRIBE customer;
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| customerId | int          | NO   | PRI | NULL    | auto_increment |
| fname      | varchar(50)  | YES  |     | NULL    |                |
| sname      | varchar(50)  | YES  |     | NULL    |                |
| address    | varchar(100) | YES  |     | NULL    |                |
| email      | varchar(100) | YES  | UNI | NULL    |                |
| phone      | varchar(15)  | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

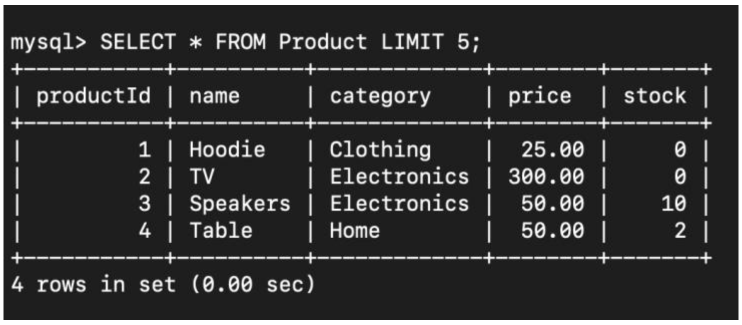


```
mysql> SELECT * FROM Customer LIMIT 5;
+------------+--------+------------+----------------+-----------------+------------+
| customerId | fname  | sname      | address        | email           | phone      |
+------------+--------+------------+----------------+-----------------+------------+
|          1 | Amelia | Hamulewicz | 17 yellow st   | ah@gmail.com    | 1112223334 |
|          2 | John   | Doe        | 66 mayfair     | jd@gmail.com    | 1122334455 |
|          4 | Ammar  | Salah      | 14 green road  | amsa@gmail.com  | 1111111122 |
|          5 | Maya   | Salah      | Gaza palestine | maysa@gmail.com | 2222222222 |
|          6 | Emma   | Walsh      | 15 blue street | emwal@gmail.com | 3333333333 |
+------------+--------+------------+----------------+-----------------+------------+
5 rows in set (0.00 sec)
```

Product Table

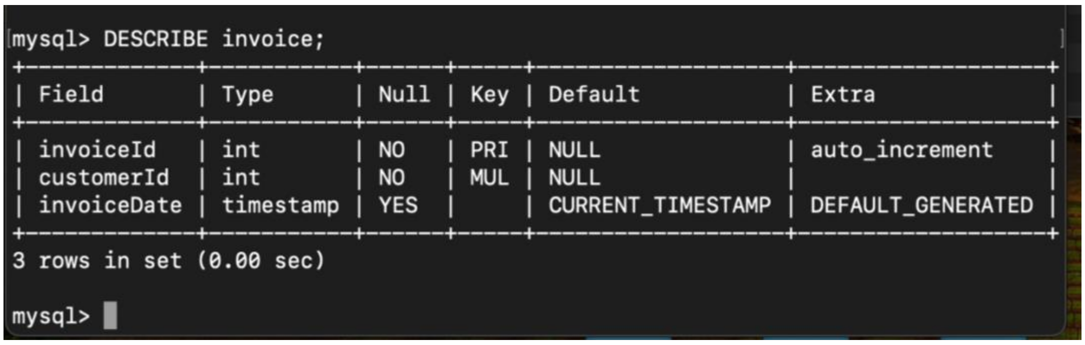


```
mysql> DESCRIBE product;
+-----------+---------------+------+-----+---------+----------------+
| Field     | Type          | Null | Key | Default | Extra          |
+-----------+---------------+------+-----+---------+----------------+
| productId | int           | NO   | PRI | NULL    | auto_increment |
| name      | varchar(100)  | YES  |     | NULL    |                |
| category  | varchar(50)   | YES  |     | NULL    |                |
| price     | decimal(10,2) | YES  |     | NULL    |                |
| stock     | int           | YES  |     | NULL    |                |
+-----------+---------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

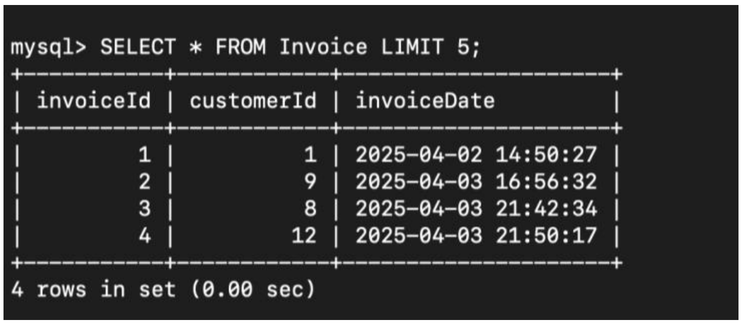


```
mysql> SELECT * FROM Product LIMIT 5;
+-----------+----------+-------------+--------+-------+
| productId | name     | category    | price  | stock |
+-----------+----------+-------------+--------+-------+
|         1 | Hoodie   | Clothing    |  25.00 |     0 |
|         2 | TV       | Electronics | 300.00 |     0 |
|         3 | Speakers | Electronics |  50.00 |    10 |
|         4 | Table    | Home        |  50.00 |     2 |
+-----------+----------+-------------+--------+-------+
4 rows in set (0.00 sec)
```

Invoice Table

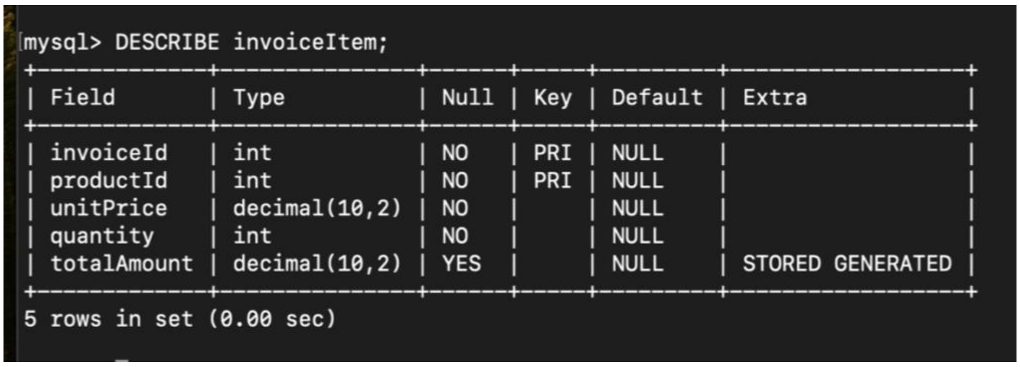


```
mysql> DESCRIBE invoice;
+-------------+-----------+------+-----+-------------------+-------------------+
| Field       | Type      | Null | Key | Default           | Extra             |
+-------------+-----------+------+-----+-------------------+-------------------+
| invoiceId   | int       | NO   | PRI | NULL              | auto_increment    |
| customerId  | int       | NO   | MUL | NULL              |                   |
| invoiceDate | timestamp | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+-------------+-----------+------+-----+-------------------+-------------------+
3 rows in set (0.00 sec)

mysql> 
```

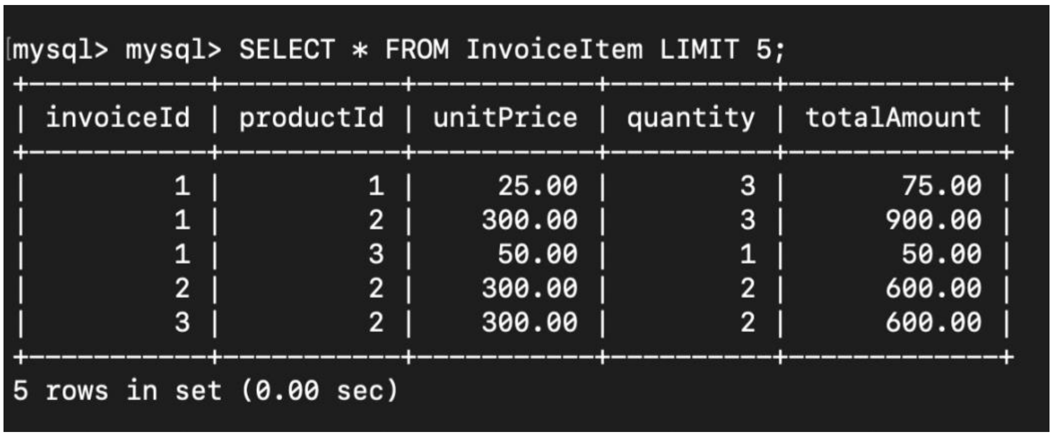


```
mysql> SELECT * FROM Invoice LIMIT 5;
+-----------+------------+---------------------+
| invoiceId | customerId | invoiceDate         |
+-----------+------------+---------------------+
|         1 |          1 | 2025-04-02 14:50:27 |
|         2 |          9 | 2025-04-03 16:56:32 |
|         3 |          8 | 2025-04-03 21:42:34 |
|         4 |         12 | 2025-04-03 21:50:17 |
+-----------+------------+---------------------+
4 rows in set (0.00 sec)
```

Invoice Item Table

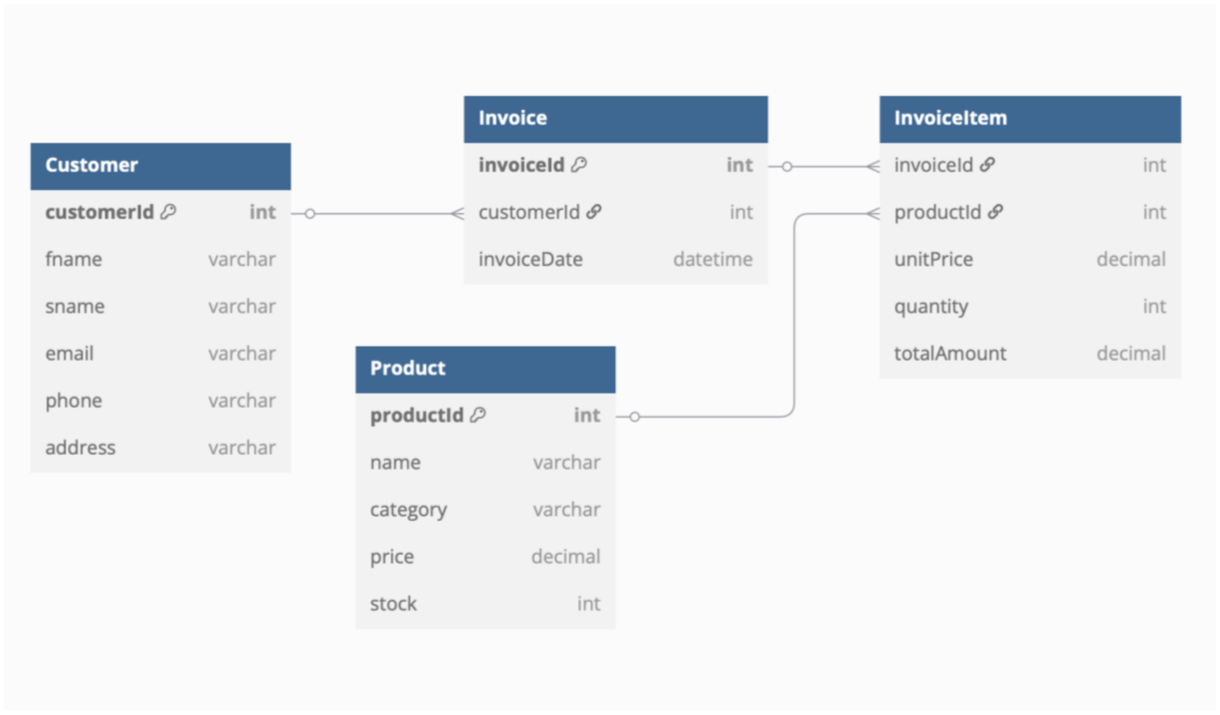


```
mysql> DESCRIBE invoiceItem;
+-------------+---------------+------+-----+---------+------------------+
| Field       | Type          | Null | Key | Default | Extra            |
+-------------+---------------+------+-----+---------+------------------+
| invoiceId   | int           | NO   | PRI | NULL    |                  |
| productId   | int           | NO   | PRI | NULL    |                  |
| unitPrice   | decimal(10,2) | NO   |     | NULL    |                  |
| quantity    | int           | NO   |     | NULL    |                  |
| totalAmount | decimal(10,2) | YES  |     | NULL    | STORED GENERATED |
+-------------+---------------+------+-----+---------+------------------+
5 rows in set (0.00 sec)
```



```
mysql> mysql> SELECT * FROM InvoiceItem LIMIT 5;
+-----------+-----------+-----------+----------+-------------+
| invoiceId | productId | unitPrice | quantity | totalAmount |
+-----------+-----------+-----------+----------+-------------+
|         1 |         1 |     25.00 |        3 |       75.00 |
|         1 |         2 |    300.00 |        3 |      900.00 |
|         1 |         3 |     50.00 |        1 |       50.00 |
|         2 |         2 |    300.00 |        2 |      600.00 |
|         3 |         2 |    300.00 |        2 |      600.00 |
+-----------+-----------+-----------+----------+-------------+
5 rows in set (0.00 sec)
```

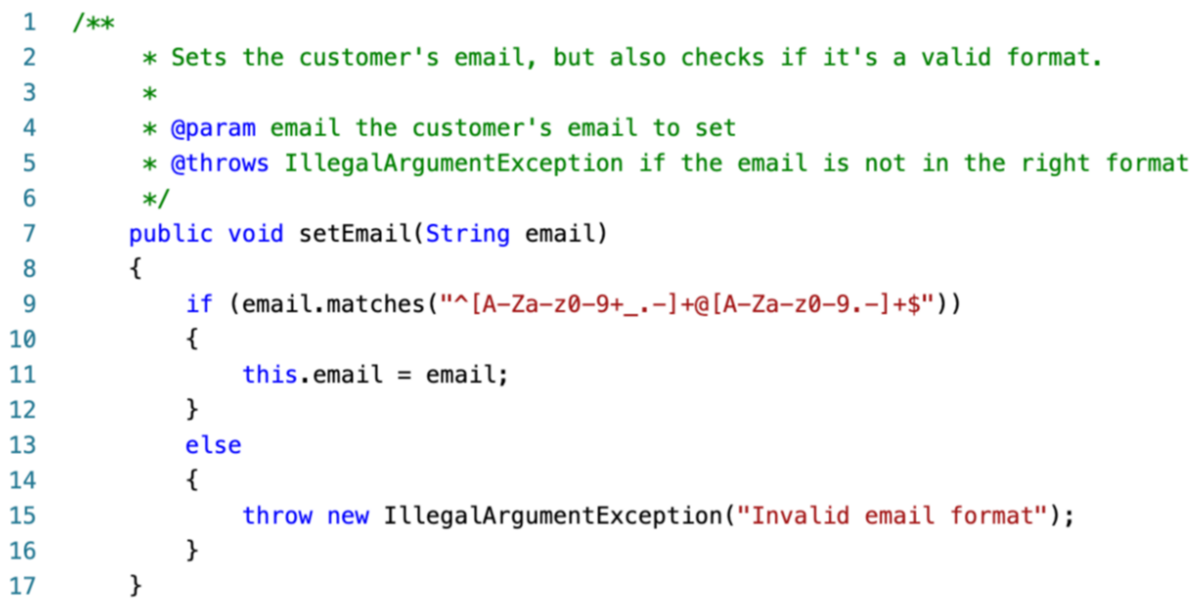
ER DIAGRAM


Customer
customerId int
fname varchar
sname varchar
email varchar
phone varchar
address varchar
Invoice
invoiceId int
customerId int
invoiceDate datetime
InvoiceItem
invoiceId int
productId int
unitPrice decimal
quantity int
totalAmount decimal
Product
productId int
name varchar
category varchar
price decimal
stock int

Interesting Source Code Snippets

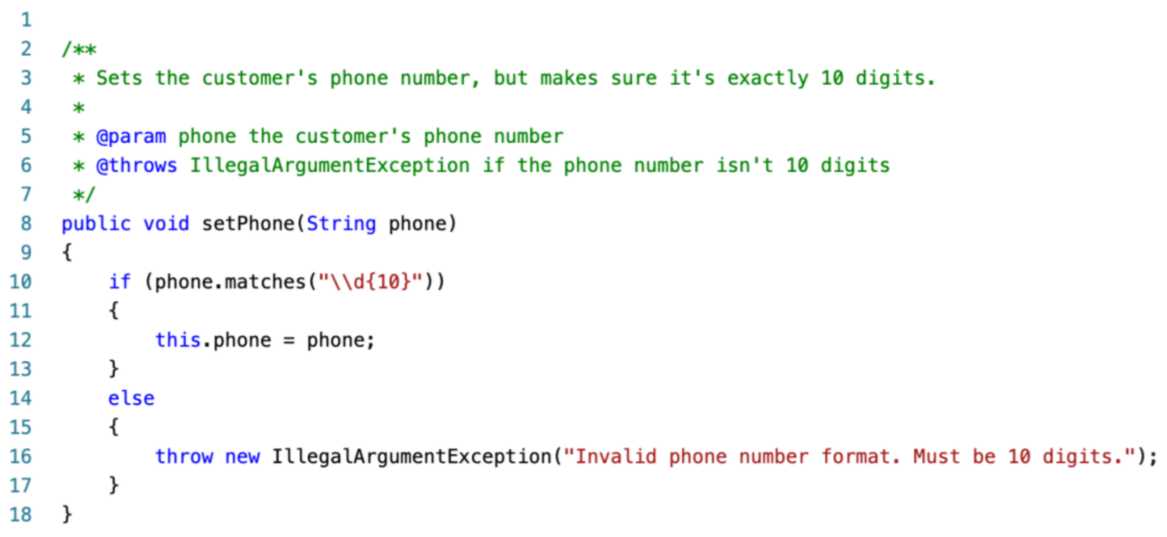
Validation

Email Validation in Customer Object



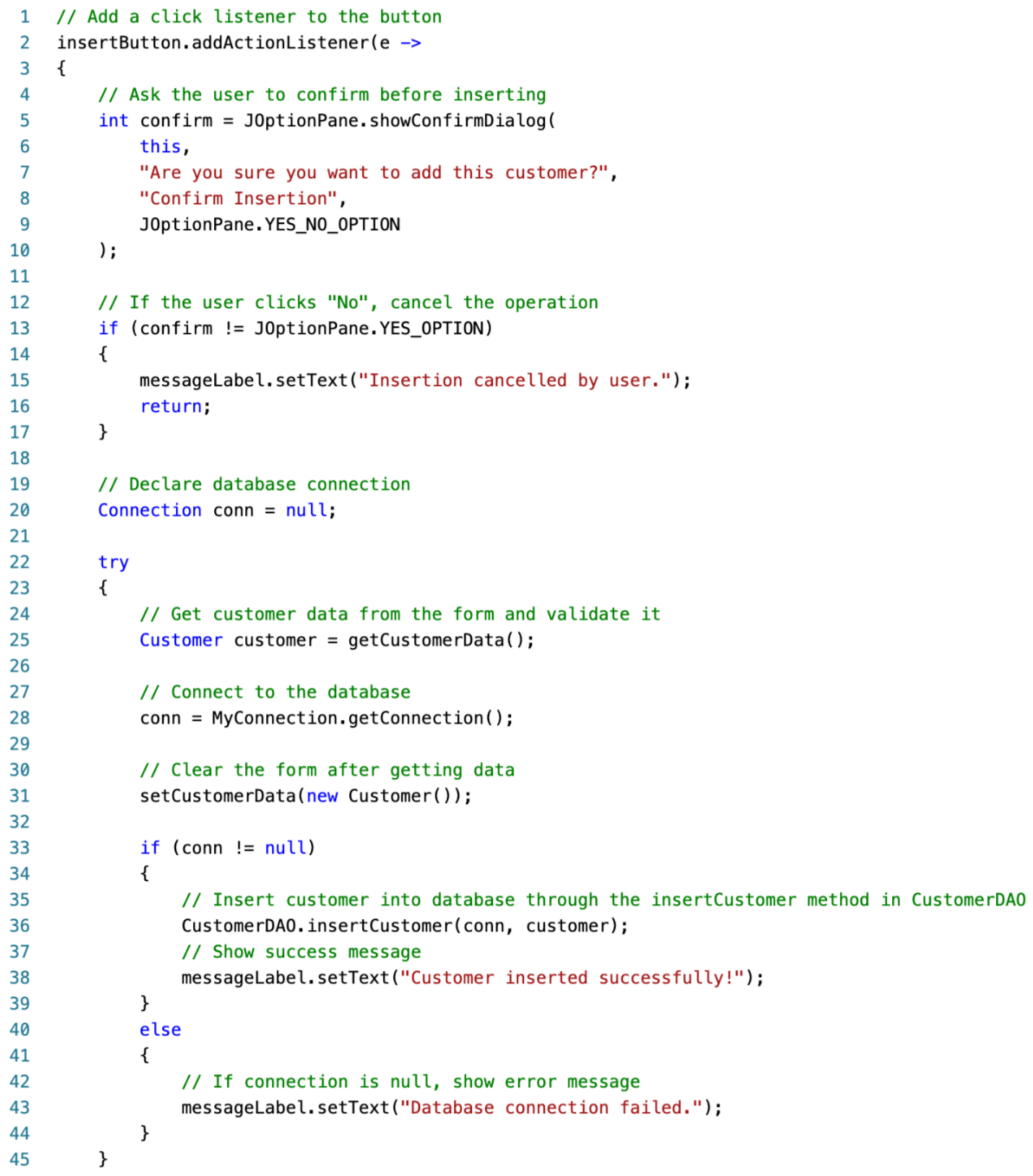
```java
/**
     * Sets the customer's email, but also checks if it's a valid format.
     *
     * @param email the customer's email to set
     * @throws IllegalArgumentException if the email is not in the right format
     */
    public void setEmail(String email)
    {
        if (email.matches("^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+$"))
        {
            this.email = email;
        }
        else
        {
            throw new IllegalArgumentException("Invalid email format");
        }
    }
```

Phone Validation in Customer Object



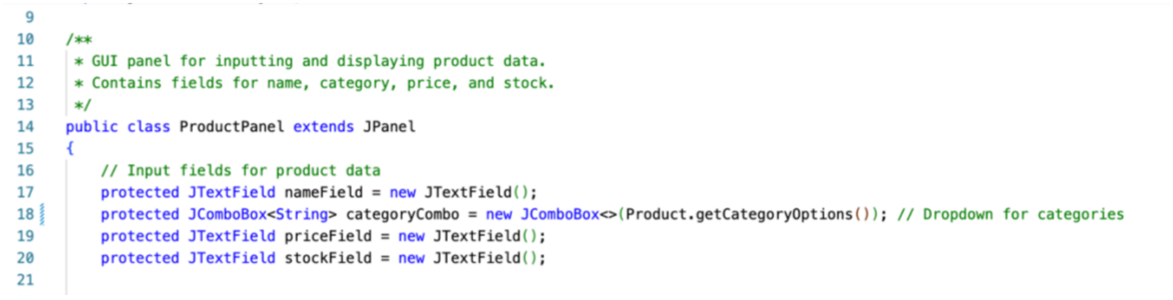
```java

/**
 * Sets the customer's phone number, but makes sure it's exactly 10 digits.
 *
 * @param phone the customer's phone number
 * @throws IllegalArgumentException if the phone number isn't 10 digits
 */
public void setPhone(String phone)
{
    if (phone.matches("\\d{10}"))
    {
        this.phone = phone;
    }
    else
    {
        throw new IllegalArgumentException("Invalid phone number format. Must be 10 digits.");
    }
}
```

Confirm Before Submission / Event handling GUI



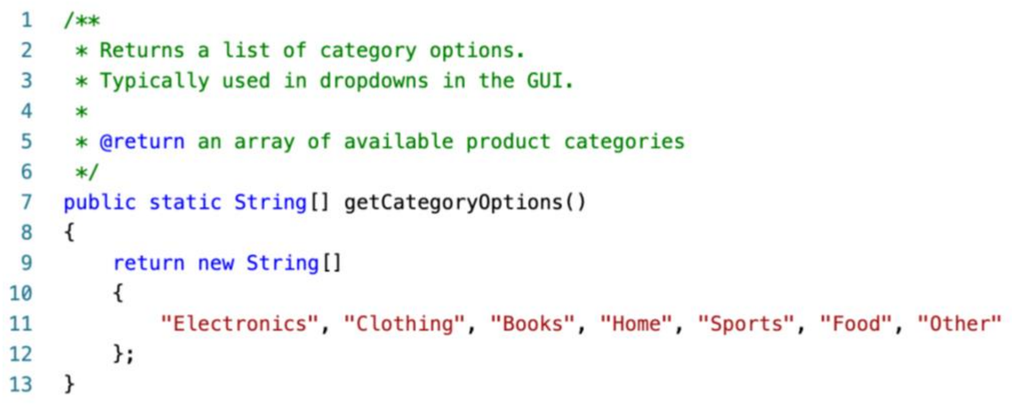
```java
// Add a click listener to the button
insertButton.addActionListener(e ->
{
    // Ask the user to confirm before inserting
    int confirm = JOptionPane.showConfirmDialog(
        this,
        "Are you sure you want to add this customer?",
        "Confirm Insertion",
        JOptionPane.YES_NO_OPTION
    );

    // If the user clicks "No", cancel the operation
    if (confirm != JOptionPane.YES_OPTION)
    {
        messageLabel.setText("Insertion cancelled by user.");
        return;
    }

    // Declare database connection
    Connection conn = null;

    try
    {
        // Get customer data from the form and validate it
        Customer customer = getCustomerData();

        // Connect to the database
        conn = MyConnection.getConnection();

        // Clear the form after getting data
        setCustomerData(new Customer());

        if (conn != null)
        {
            // Insert customer into database through the insertCustomer method in CustomerDAO
            CustomerDAO.insertCustomer(conn, customer);
            // Show success message
            messageLabel.setText("Customer inserted successfully!");
        }
        else
        {
            // If connection is null, show error message
            messageLabel.setText("Database connection failed.");
        }
    }
```

Product Categories list dropdown



```java

/**
 * GUI panel for inputting and displaying product data.
 * Contains fields for name, category, price, and stock.
 */
public class ProductPanel extends JPanel
{
    // Input fields for product data
    protected JTextField nameField = new JTextField();
    protected JComboBox<String> categoryCombo = new JComboBox<>(Product.getCategoryOptions()); // Dropdown for categories
    protected JTextField priceField = new JTextField();
    protected JTextField stockField = new JTextField();

```

Default categories



```java
/**
 * Returns a list of category options.
 * Typically used in dropdowns in the GUI.
 *
 * @return an array of available product categories
 */
public static String[] getCategoryOptions()
{
    return new String[]
    {
        "Electronics", "Clothing", "Books", "Home", "Sports", "Food", "Other"
    };
}
```

Updating stock in the product table when item/s are added into invoice table:

Invoice DAO

```java
/**
  * Updates the quantity of a product in an invoice and adjusts stock accordingly.
  *
  * @param invoiceId the invoice ID
  * @param productId the product ID
  * @param newQty    the new quantity
  * @return true if the quantity was updated, false otherwise
  * @throws SQLException if there is not enough stock or a database error occurs
  */
 public boolean updateInvoiceItemQuantity(int invoiceId, int productId, int newQty) throws SQLException {
     String getQtySQL = "SELECT quantity FROM InvoiceItem WHERE invoiceId = ? AND productId = ?";
     int oldQty;

     try (PreparedStatement getStmt = conn.prepareStatement(getQtySQL)) {
         getStmt.setInt(1, invoiceId);
         getStmt.setInt(2, productId);
         ResultSet rs = getStmt.executeQuery();
         if (rs.next()) {
             oldQty = rs.getInt("quantity");
         } else {
             throw new SQLException("Invoice item not found.");
         }
     }
```

Add/ amend invoice items...

```java
private void updateQuantity()
    {
        if (selectedInvoiceId == -1 || selectedProductId == -1)
        {
            messageLabel.setText("Select an invoice and item first.");
            return;
        }

        try (Connection conn = MyConnection.getConnection())
        {
            int newQty = Integer.parseInt(quantityField.getText().trim());
            InvoiceDAO dao = new InvoiceDAO(conn);
            boolean updated = dao.updateInvoiceItemQuantity(selectedInvoiceId, selectedProductId, newQty);

            if (updated)
            {
                messageLabel.setText("Quantity updated.");
                ViewInvoiceTable.loadAll(invoiceModel);
                loadInvoiceItems(selectedInvoiceId);
            }
            else
            {
                messageLabel.setText("Update failed.");
            }
        }
        catch (Exception ex)
        {
            messageLabel.setText("Error: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
```

Tests

1. **Product Management**

   Test: Add a new product
   - Entered valid name, category, price, and stock.
   - Clicked 'Insert Product'.
   - Product was saved to the database and showed up in the table.

   Test: Add product with negative price
   - Entered -5 for price.
   - System showed a validation error: 'Price cannot be negative'.

   Test: Delete a product
   - Selected a product from the table.
   - Clicked 'Delete Product'.
   - Product was removed and table updated.

2. **Customer Management**

   Test: Add new customer
   - Entered valid name, email, and phone.
   - Customer was saved and displayed correctly.

   Test: Add customer with duplicate email
   - Entered same email as existing customer.
   - System showed SQL error for duplicate email (as expected).

   Test: Edit customer
   - Selected a customer, changed details, and saved.
   - Changes saved and updated in the table.

3. **Invoice Management**

   Test: Create invoice with items
   - Selected a customer, added items from products.
   - Invoice was created and total calculated.

   Test: Add item with too much quantity
   - Tried to add item with quantity higher than stock.
   - System blocked it and showed 'Not enough stock'.

   Test: Delete an invoice
   - Selected invoice and clicked delete.
   - Invoice and items were removed.

4. **Input Validation & Error Handling**
   Test: Enter letters in price field
   - Typed 'abc' instead of a number.
   - Error message shown: 'Please enter valid numbers'.

   Test: Leave fields empty
   - Tried to submit form without filling required fields.
   - App showed validation messages and didn't crash.

5. **Filtering and Viewing**
   Test: Filter products by category
   - Selected 'Clothing' from category filter.
   - Only clothing products shown.

   Test: View all invoices
   - Loaded invoice