Even Semester (2023)

Assignment Cover Letter

(Group Assignment)

Student Names: Davin Neilson	**Student IDs:** 2602119190 2602119303 2602109870
Course Code: COMP6699001	**Course Name:** Object Oriented Programming
Class: L2AC	**Name of Lecturer:** Jude Joseph Lamug Martinez, MCS
Major: Computer Science	
Title of Assignment: Jewelry Management System	
Type of Assignment: Final Project	
Due Date: 16 - 06 - 2023	

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signatures of Students:

Davin Neilson

[illegible]

Table of Contents

I. Overview

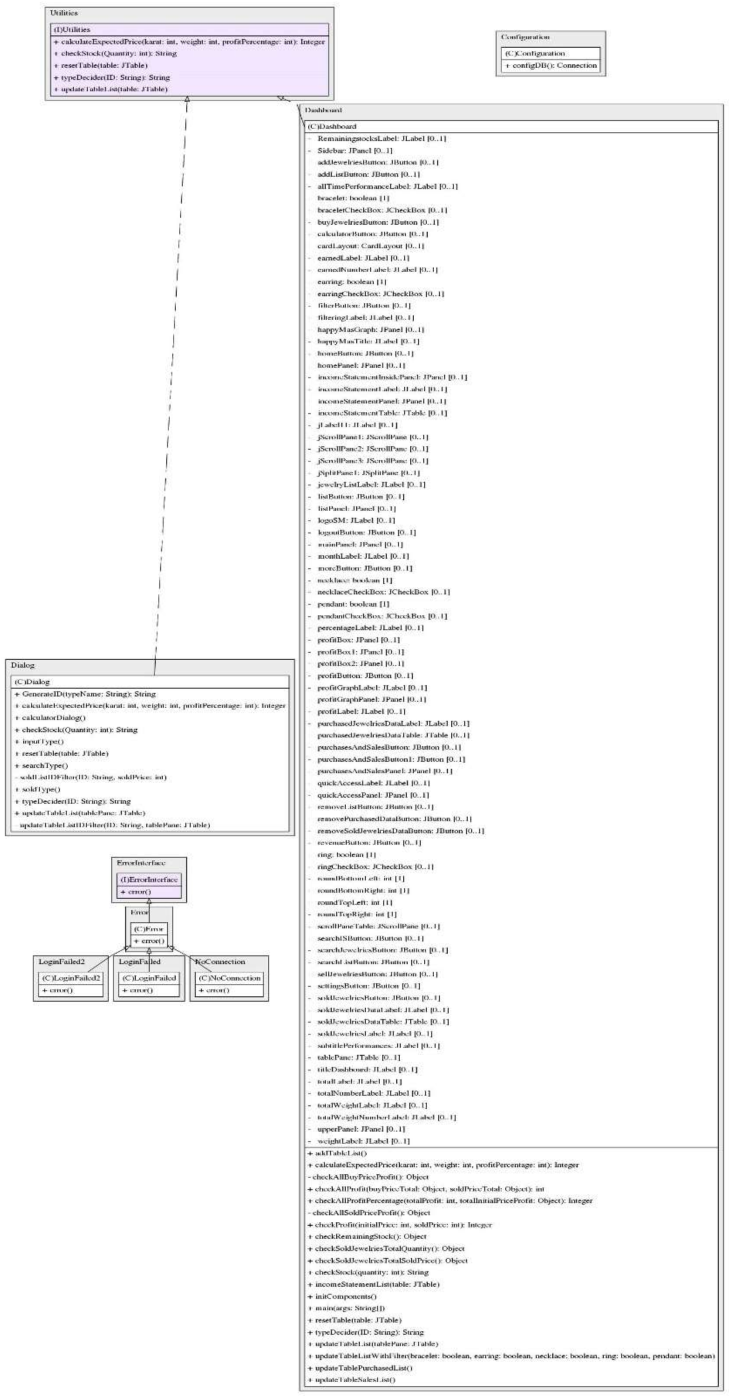
A. Background

Before deciding on a final project for the Object-Oriented Programming course, there were many ideas that popped up. However, there was one idea that really caught my attention. At that point, my mom was telling me about her jewelry store problem. She needs a program that can store data that is important for her store, such as income statements. Other than that, she also wishes that she were able to search the data by inputting the code of the products only. Therefore, a jewelry management system may be a good idea that will be implemented for my Object-Oriented Programming final project.

B. Description

The jewelry management system is a program where the user can access the jewelry data by only inputting the product code. Other than that, the user is also able to do these activities, such as creating a jewelry type, determining the sold price for each piece of jewelry, and also observing the income statement, jewelry list, purchased jewelry list, and sold jewelry list. Moreover, in the dashboard, the user can see how well the jewelry shop's performance is.

II. Class Diagram


Utilities
(I)Utilities
+ calculateExpectedPrice(karat: int, weight: int, profitPercentage: int): Integer
+ checkStock(Quantity: int): String
+ resetTable(table: JTable)
+ typeDecider(ID: String): String
+ updateTableList(table: JTable)
Configuration
(C)Configuration
+ configDB(): Connection
Dashboard
(C)Dashboard
- RemainingstocksLabel: JLabel [0..1]
- Sidebar: JPanel [0..1]
- addJewelriesButton: JButton [0..1]
- addListButton: JButton [0..1]
- allTimePerformanceLabel: JLabel [0..1]
- bracelet: boolean [1]
- braceletCheckBox: JCheckBox [0..1]
- buyJewelriesButton: JButton [0..1]
- calculatorButton: JButton [0..1]
- cardLayout: CardLayout [0..1]
- earnedLabel: JLabel [0..1]
- earnedNumberLabel: JLabel [0..1]
- earring: boolean [1]
- earringCheckBox: JCheckBox [0..1]
- filterButton: JButton [0..1]
- filteringLabel: JLabel [0..1]
- happyMasGraph: JPanel [0..1]
- happyMasTitle: JLabel [0..1]
- homeButton: JButton [0..1]
- homePanel: JPanel [0..1]
- incomeStatementInsidePanel: JPanel [0..1]
- incomeStatementLabel: JLabel [0..1]
- incomeStatementPanel: JPanel [0..1]
- incomeStatementTable: JTable [0..1]
- jLabel11: JLabel [0..1]
- jScrollPane1: JScrollPane [0..1]
- jScrollPane2: JScrollPane [0..1]
- jScrollPane3: JScrollPane [0..1]
- jSplitPane1: JSplitPane [0..1]
- jewelryListLabel: JLabel [0..1]
- listButton: JButton [0..1]
- listPanel: JPanel [0..1]
- logoSM: JLabel [0..1]
- logoutButton: JButton [0..1]
- mainPanel: JPanel [0..1]
- monthLabel: JLabel [0..1]
- moreButton: JButton [0..1]
- necklace: boolean [1]
- necklaceCheckBox: JCheckBox [0..1]
- pendant: boolean [1]
- pendantCheckBox: JCheckBox [0..1]
- percentageLabel: JLabel [0..1]
- profitBox: JPanel [0..1]
- profitBox1: JPanel [0..1]
- profitBox2: JPanel [0..1]
- profitButton: JButton [0..1]
- profitGraphLabel: JLabel [0..1]
- profitGraphPanel: JPanel [0..1]
- profitLabel: JLabel [0..1]
- purchasedJewelriesDataLabel: JLabel [0..1]
- purchasedJewelriesDataTable: JTable [0..1]
- purchasesAndSalesButton: JButton [0..1]
- purchasesAndSalesButton1: JButton [0..1]
- purchasesAndSalesPanel: JPanel [0..1]
- quickAccessLabel: JLabel [0..1]
- quickAccessPanel: JPanel [0..1]
- removeListButton: JButton [0..1]
- removePurchasedDataButton: JButton [0..1]
- removeSoldJewelriesDataButton: JButton [0..1]
- revenueButton: JButton [0..1]
- ring: boolean [1]
- ringCheckBox: JCheckBox [0..1]
- roundBottomLeft: int [1]
- roundBottomRight: int [1]
- roundTopLeft: int [1]
- roundTopRight: int [1]
- scrollPaneTable: JScrollPane [0..1]
- searchISButton: JButton [0..1]
- searchJewelriesButton: JButton [0..1]
- searchListButton: JButton [0..1]
- sellJewelriesButton: JButton [0..1]
- settingsButton: JButton [0..1]
- soldJewelriesButton: JButton [0..1]
- soldJewelriesDataLabel: JLabel [0..1]
- soldJewelriesDataTable: JTable [0..1]
- soldJewelriesLabel: JLabel [0..1]
- subtitlePerformances: JLabel [0..1]
- tablePane: JTable [0..1]
- titleDashboard: JLabel [0..1]
- totalLabel: JLabel [0..1]
- totalNumberLabel: JLabel [0..1]
- totalWeightLabel: JLabel [0..1]
- totalWeightNumberLabel: JLabel [0..1]
- upperPanel: JPanel [0..1]
- weightLabel: JLabel [0..1]
+ addTableList()
+ calculateExpectedPrice(karat: int, weight: int, profitPercentage: int): Integer
- checkAllBuyPriceProfit(): Object
+ checkAllProfit(buyPriceTotal: Object, soldPriceTotal: Object): int
+ checkAllProfitPercentage(totalProfit: int, totalInitialPriceProfit: Object): Integer
- checkAllSoldPriceProfit(): Object
+ checkProfit(initialPrice: int, soldPrice: int): Integer
+ checkRemainingStock(): Object
+ checkSoldJewelriesTotalQuantity(): Object
+ checkSoldJewelriesTotalSoldPrice(): Object
+ checkStock(quantity: int): String
+ incomeStatementList(table: JTable)
+ initComponents()
+ main(args: String[])
+ resetTable(table: JTable)
+ typeDecider(ID: String): String
+ updateTableList(tablePane: JTable)
+ updateTableListWithFilter(bracelet: boolean, earring: boolean, necklace: boolean, ring: boolean, pendant: boolean)
+ updateTablePurchasedList()
+ updateTableSalesList()
Dialog
(C)Dialog
+ GenerateID(typeName: String): String
+ calculateExpectedPrice(karat: int, weight: int, profitPercentage: int): Integer
+ calculatorDialog()
+ checkStock(Quantity: int): String
+ inputType()
+ resetTable(table: JTable)
+ searchType()
- soldListIDFilter(ID: String, soldPrice: int)
+ soldType()
+ typeDecider(ID: String): String
+ updateTableList(tablePane: JTable)
- updateTableListIDFilter(ID: String, tablePane: JTable)
ErrorInterface
(I)ErrorInterface
+ error()
Error
(C)Error
+ error()
LoginFailed2
(C)LoginFailed2
+ error()
LoginFailed
(C)LoginFailed
+ error()
NoConnection
(C)NoConnection
+ error()

III. **Modules/Library**

There are some modules and libraries that are implemented for this program, they are:

A. Java Swing

   Java Swing is a GUI toolkit where it allows creators to create visual and interactive applications. There are many types of java swing components such as buttons, text-fields, password fields, panels, and also menus. These components can be customized easily and offer various features for each component. Moreover, many IDEs already provide a Java swing customization where the user can just use drag and drop features to create the UI of the programs. Thus, it will generate the code after creating the UI.

B. SQL Connector / JDBC (Java Database Connectivity)

   JDBC is one type of Java API, where it can interact with databases using SQL. It also allows java applications to do certain query for SQL such as creating, updating the database, retrieve the results and many more.

C. ArrayList

   ArrayList is one type of Java API where it provides an implementation of the List interface and represents a resizable array or dynamic array. They provide many methods to add, remove, access, and modify elements in a specific position in an array.

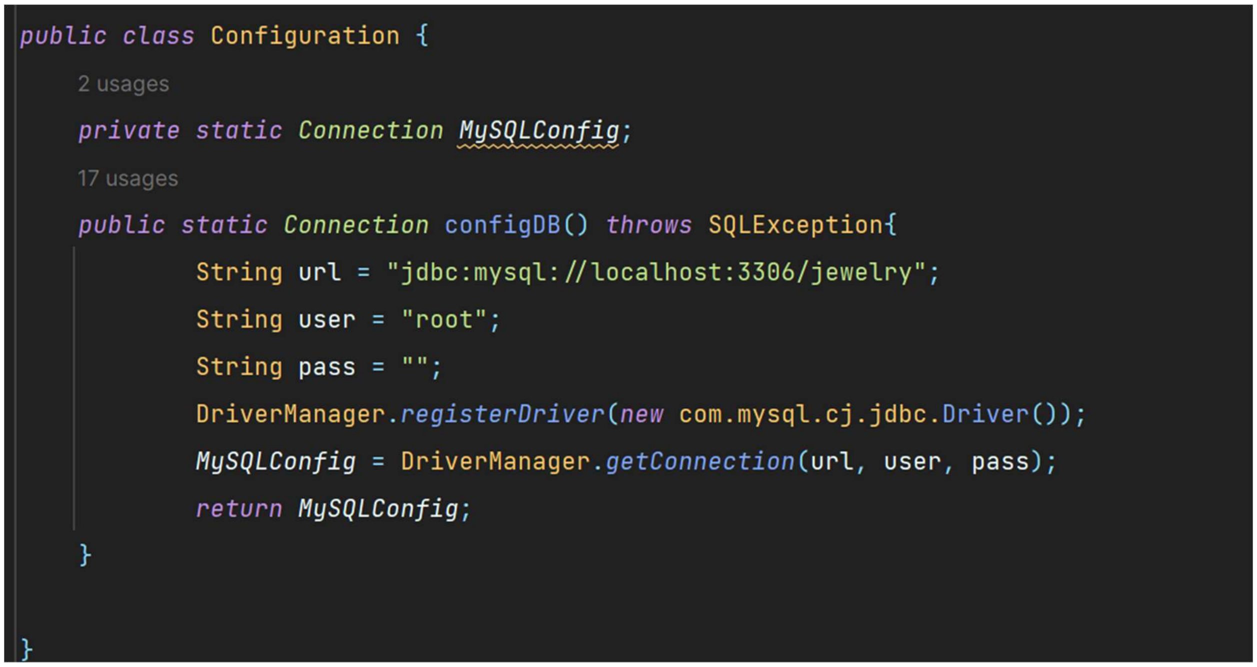
D. FlatLaf

   Flatlaf is a part of java swing, where it can change the theme or look and feel of a java frame better. There are a lot of themes that are provided in FlatLaf, such as FlatLaf Dark, FlatLaf Light, and others.

IV. Implementation

There are six classes in total in this program. Therefore, there will be six main points that will be discussed in the implementation part.

A. Configuration



```
public class Configuration {
    2 usages
    private static Connection MySQLConfig;
    17 usages
    public static Connection configDB() throws SQLException{
            String url = "jdbc:mysql://localhost:3306/jewelry";
            String user = "root";
            String pass = "";
            DriverManager.registerDriver(new com.mysql.cj.jdbc.Driver());
            MySQLConfig = DriverManager.getConnection(url, user, pass);
            return MySQLConfig;
    }

}
```

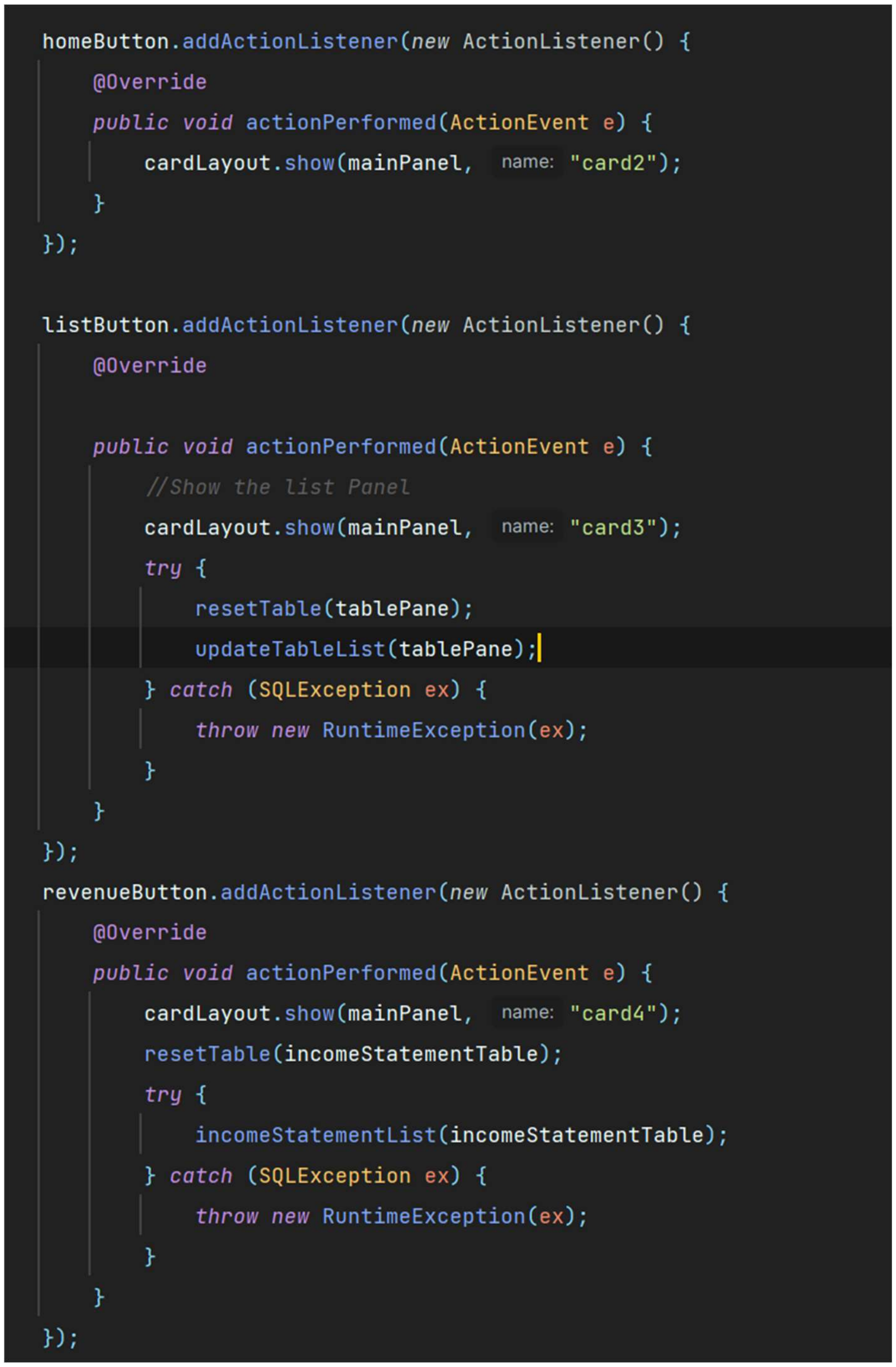
For this class, there are the basic settings for the SQL Connection, so whenever the developer calls the class. It will return the Connection for SQL Database.

B. Dashboard

There are many methods and variables that are implemented in this class:

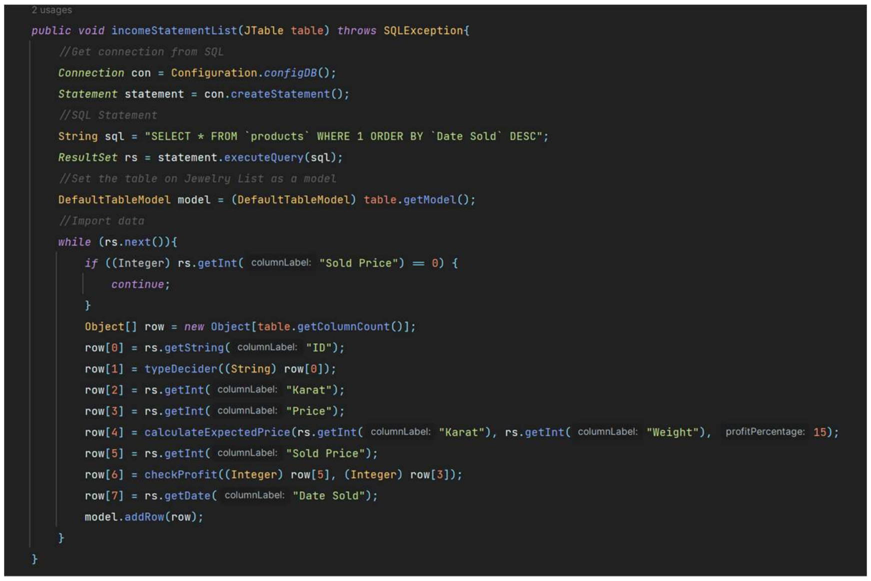
1. initComponents()

   Since the dashboard class is extended to the Java frame, the initComponents() will make the program visible and put the important components to the window. Other than that, this method will give each button an action listener, where they have their own commands, such as the sidebar. The sidebar contains four buttons, they are dashboard, jewelry list, income statement, and log out buttons. The dashboard will show the dashboard page, the jewelry list will show its page and it is also the same as the income statement. However, the logout button will dispose of the frame and call the LoginMenu class to show the login menu. The picture below shows how the sidebar works and how I implemented the action listener to these buttons.



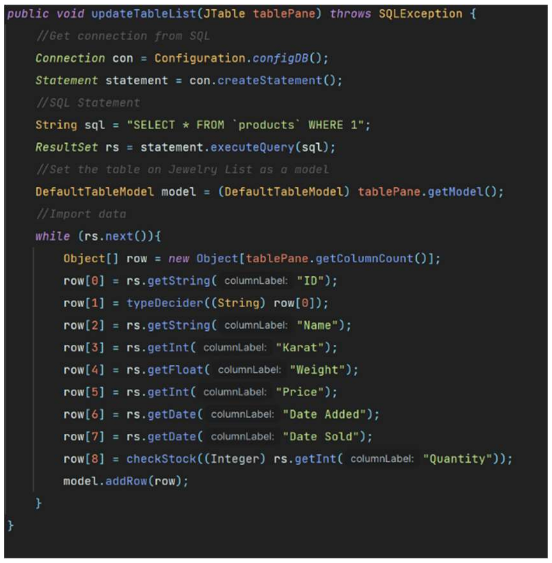
```
homeButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        cardLayout.show(mainPanel,  name: "card2");
    }
});

listButton.addActionListener(new ActionListener() {
    @Override

    public void actionPerformed(ActionEvent e) {
        //Show the list Panel
        cardLayout.show(mainPanel,  name: "card3");
        try {
            resetTable(tablePane);
            updateTableList(tablePane);
        } catch (SQLException ex) {
            throw new RuntimeException(ex);
        }
    }
});
revenueButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        cardLayout.show(mainPanel,  name: "card4");
        resetTable(incomeStatementTable);
        try {
            incomeStatementList(incomeStatementTable);
        } catch (SQLException ex) {
            throw new RuntimeException(ex);
        }
    }
});
```

2. IncomeStatementList() & updateTableList()



```java
2 usages
public void incomeStatementList(JTable table) throws SQLException{
    //Get connection from SQL
    Connection con = Configuration.configDB();
    Statement statement = con.createStatement();
    //SQL Statement
    String sql = "SELECT * FROM `products` WHERE 1 ORDER BY `Date Sold` DESC";
    ResultSet rs = statement.executeQuery(sql);
    //Set the table on Jewelry List as a model
    DefaultTableModel model = (DefaultTableModel) table.getModel();
    //Import data
    while (rs.next()){
        if ((Integer) rs.getInt( columnLabel: "Sold Price") == 0) {
            continue;
        }
        Object[] row = new Object[table.getColumnCount()];
        row[0] = rs.getString( columnLabel: "ID");
        row[1] = typeDecider((String) row[0]);
        row[2] = rs.getInt( columnLabel: "Karat");
        row[3] = rs.getInt( columnLabel: "Price");
        row[4] = calculateExpectedPrice(rs.getInt( columnLabel: "Karat"), rs.getInt( columnLabel: "Weight"),  profitPercentage: 15);
        row[5] = rs.getInt( columnLabel: "Sold Price");
        row[6] = checkProfit((Integer) row[5], (Integer) row[3]);
        row[7] = rs.getDate( columnLabel: "Date Sold");
        model.addRow(row);
    }
}
```

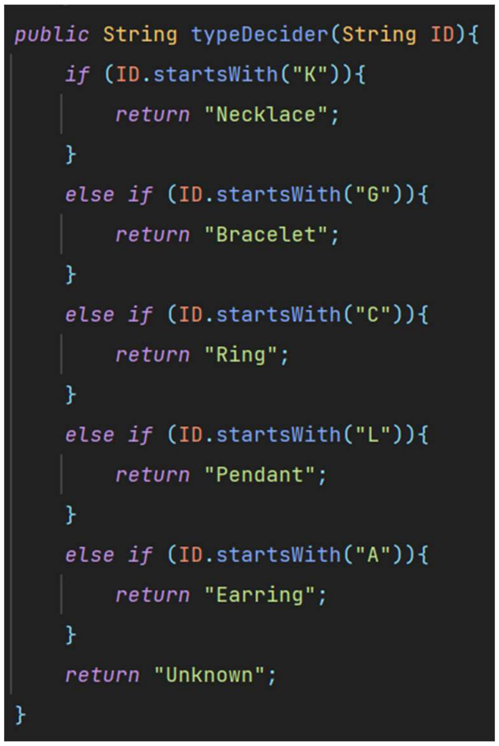
This method will be used for income statement page. In here, the developer call the Configuration class to get the Connection from the SQL and creating a statement for the result set. Then, the developer creates the SQL query where it selects any products and orders them by the 'Date Sold' column in descending order. After that, we can execute the query and set the desired table as the model. Lastly, we import the data from the database to the table until there is no other data available.



```java
public void updateTableList(JTable tablePane) throws SQLException {
    //Get connection from SQL
    Connection con = Configuration.configDB();
    Statement statement = con.createStatement();
    //SQL Statement
    String sql = "SELECT * FROM `products` WHERE 1";
    ResultSet rs = statement.executeQuery(sql);
    //Set the table on Jewelry List as a model
    DefaultTableModel model = (DefaultTableModel) tablePane.getModel();
    //Import data
    while (rs.next()){
        Object[] row = new Object[tablePane.getColumnCount()];
        row[0] = rs.getString( columnLabel: "ID");
        row[1] = typeDecider((String) row[0]);
        row[2] = rs.getString( columnLabel: "Name");
        row[3] = rs.getInt( columnLabel: "Karat");
        row[4] = rs.getFloat( columnLabel: "Weight");
        row[5] = rs.getInt( columnLabel: "Price");
        row[6] = rs.getDate( columnLabel: "Date Added");
        row[7] = rs.getDate( columnLabel: "Date Sold");
        row[8] = checkStock((Integer) rs.getInt( columnLabel: "Quantity"));
        model.addRow(row);
    }
}
```

For updateTableList method is almost the same as the previous one, however it will only import any products without any conditions.

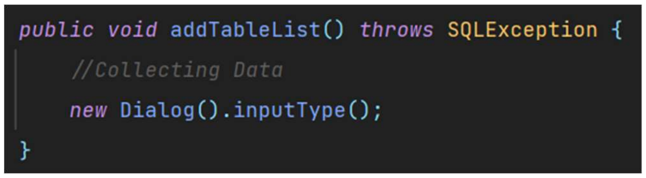
3. typeDecider()



```
public String typeDecider(String ID){
    if (ID.startsWith("K")){
        return "Necklace";
    }
    else if (ID.startsWith("G")){
        return "Bracelet";
    }
    else if (ID.startsWith("C")){
        return "Ring";
    }
    else if (ID.startsWith("L")){
        return "Pendant";
    }
    else if (ID.startsWith("A")){
        return "Earring";
    }
    return "Unknown";
}
```

For this method, it is pretty straightforward. When the String ID starts with "K", it will return the "Necklace" string. Similarly, when it starts with "G", it returns the string of "Bracelet", and so on.

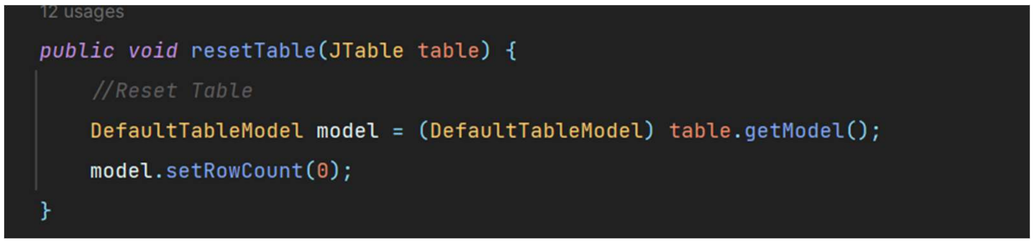
4. addTableList()



```
public void addTableList() throws SQLException {
    //Collecting Data
    new Dialog().inputType();
}
```

This method only calls the inputType method from the dialog class which will be explained later. This is implemented on the add button in every page.

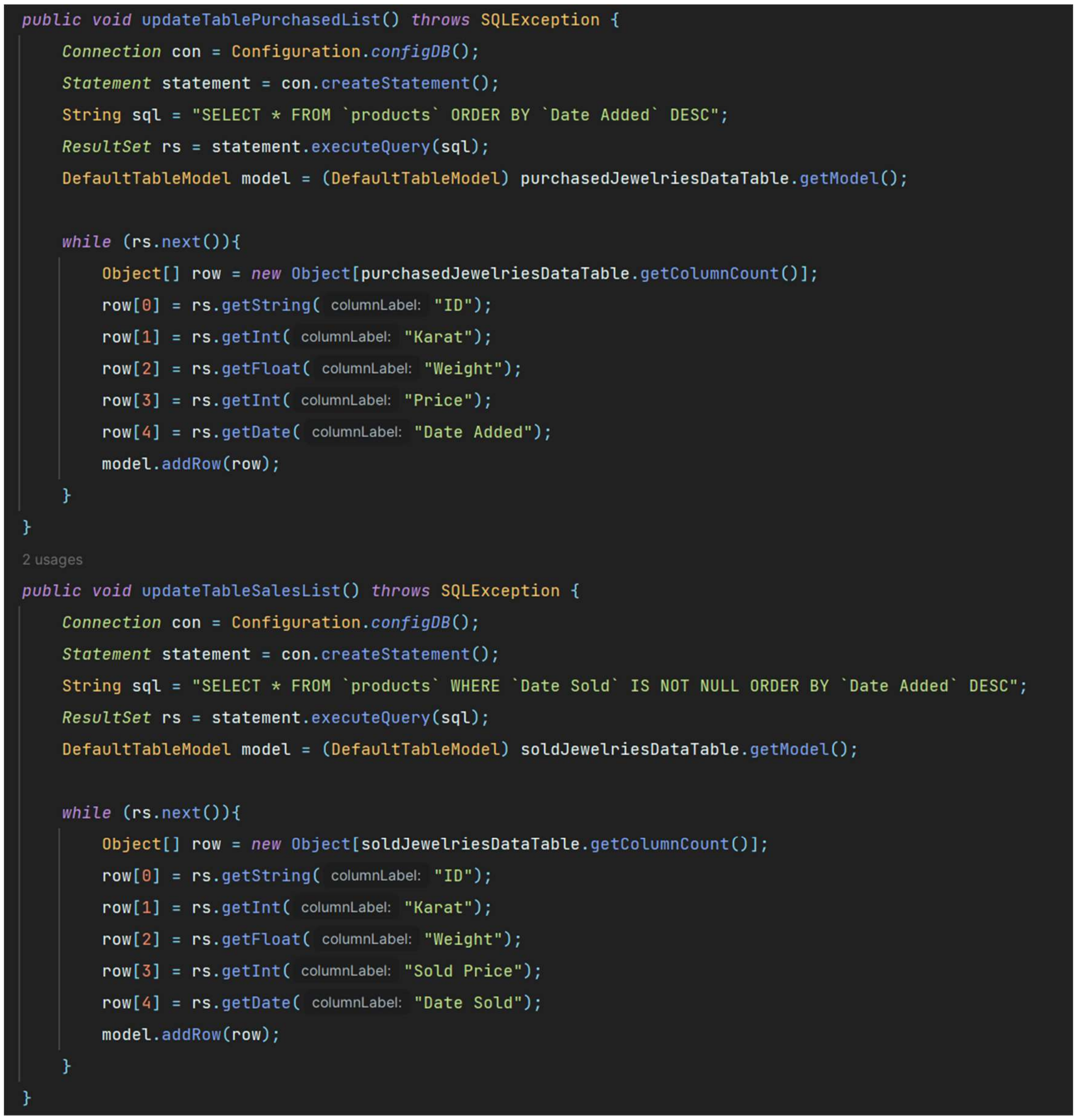
5. resetTable()



```
12 usages
public void resetTable(JTable table) {
    //Reset Table
    DefaultTableModel model = (DefaultTableModel) table.getModel();
    model.setRowCount(0);
}
```

This method will get the model from a desired table and set the row count to 0 which will clear everything from the table. This can be used in any table.

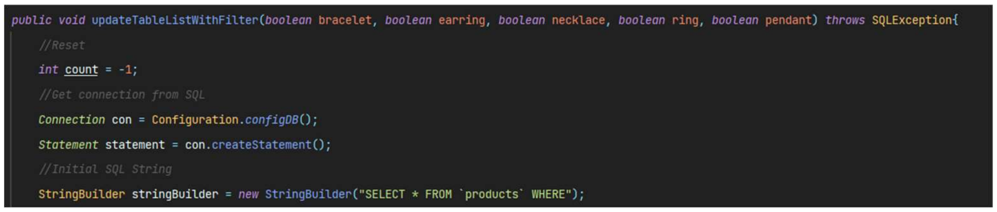
6. updateTablePurchasedList() and updateTableSalesList()



```
public void updateTablePurchasedList() throws SQLException {
    Connection con = Configuration.configDB();
    Statement statement = con.createStatement();
    String sql = "SELECT * FROM `products` ORDER BY `Date Added` DESC";
    ResultSet rs = statement.executeQuery(sql);
    DefaultTableModel model = (DefaultTableModel) purchasedJewelriesDataTable.getModel();

    while (rs.next()){
        Object[] row = new Object[purchasedJewelriesDataTable.getColumnCount()];
        row[0] = rs.getString( columnLabel: "ID");
        row[1] = rs.getInt( columnLabel: "Karat");
        row[2] = rs.getFloat( columnLabel: "Weight");
        row[3] = rs.getInt( columnLabel: "Price");
        row[4] = rs.getDate( columnLabel: "Date Added");
        model.addRow(row);
    }
}
2 usages
public void updateTableSalesList() throws SQLException {
    Connection con = Configuration.configDB();
    Statement statement = con.createStatement();
    String sql = "SELECT * FROM `products` WHERE `Date Sold` IS NOT NULL ORDER BY `Date Added` DESC";
    ResultSet rs = statement.executeQuery(sql);
    DefaultTableModel model = (DefaultTableModel) soldJewelriesDataTable.getModel();

    while (rs.next()){
        Object[] row = new Object[soldJewelriesDataTable.getColumnCount()];
        row[0] = rs.getString( columnLabel: "ID");
        row[1] = rs.getInt( columnLabel: "Karat");
        row[2] = rs.getFloat( columnLabel: "Weight");
        row[3] = rs.getInt( columnLabel: "Sold Price");
        row[4] = rs.getDate( columnLabel: "Date Sold");
        model.addRow(row);
    }
}
```

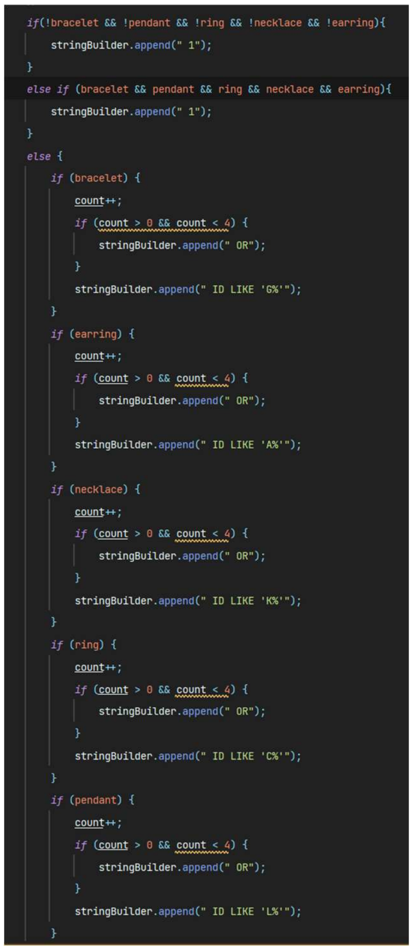
For both of these methods it is almost the same as the incomeStatementList() method where both of them will get the SQL Connection and create the connection statement then import the data from the database to the table. However, their SQL query is different. For the purchased jewelry list, the SQL will select any product and order it by ‘Date Added’ columns in Descending order, while updateTableSalesList() will select any products where the ‘Data Sold’ columns are not empty and order them by ‘Date Added’ in descending order. This will be used for updating table purchased list in purchases and sales page.

7. updateTableListWithFilter()



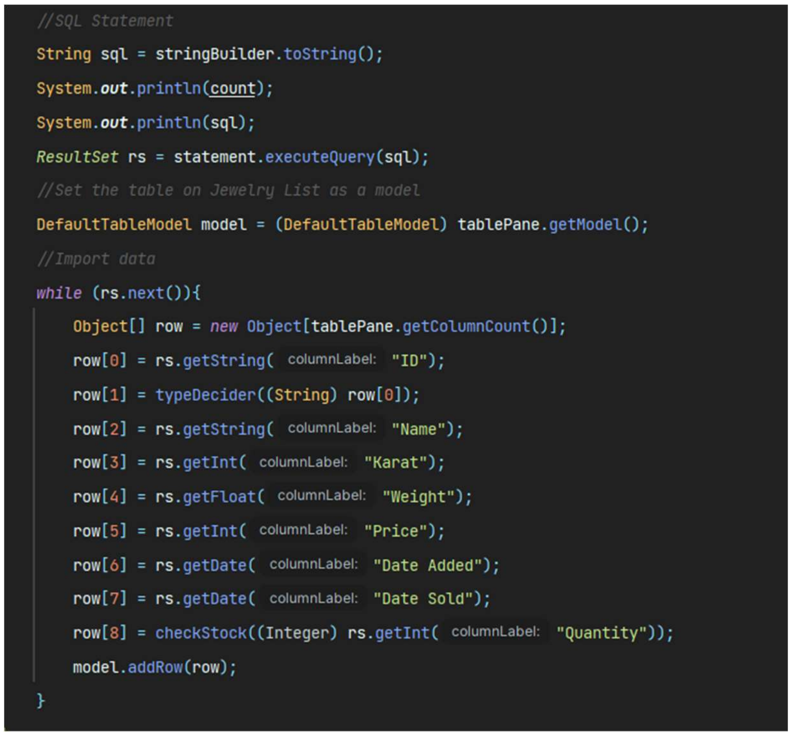
```java
public void updateTableListWithFilter(boolean bracelet, boolean earring, boolean necklace, boolean ring, boolean pendant) throws SQLException{
    //Reset
    int count = -1;
    //Get connection from SQL
    Connection con = Configuration.configDB();
    Statement statement = con.createStatement();
    //Initial SQL String
    StringBuilder stringBuilder = new StringBuilder("SELECT * FROM `products` WHERE");
```

For this part, it is quite unique because this will filter the table by using a string builder. Initially, every type of jewelry has a false value. Firstly, I will set the count as -1, connection of the SQL and create the statement as normal. Then, I call the StringBuilder class and set the initial string where it means select any products. In here, we can see that the SQL query is incomplete. Therefore, the next picture will show the condition.



```java
if(!bracelet && !pendant && !ring && !necklace && !earring){
    stringBuilder.append(" 1");
}
else if (bracelet && pendant && ring && necklace && earring){
    stringBuilder.append(" 1");
}
else {
    if (bracelet) {
        count++;
        if (count > 0 && count < 4) {
            stringBuilder.append(" OR");
        }
        stringBuilder.append(" ID LIKE 'G%'");
    }
    if (earring) {
        count++;
        if (count > 0 && count < 4) {
            stringBuilder.append(" OR");
        }
        stringBuilder.append(" ID LIKE 'A%'");
    }
    if (necklace) {
        count++;
        if (count > 0 && count < 4) {
            stringBuilder.append(" OR");
        }
        stringBuilder.append(" ID LIKE 'K%'");
    }
    if (ring) {
        count++;
        if (count > 0 && count < 4) {
            stringBuilder.append(" OR");
        }
        stringBuilder.append(" ID LIKE 'C%'");
    }
    if (pendant) {
        count++;
        if (count > 0 && count < 4) {
            stringBuilder.append(" OR");
        }
        stringBuilder.append(" ID LIKE 'L%'");
    }
```

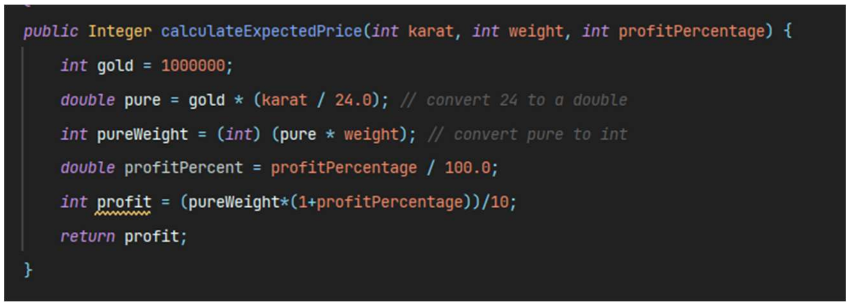
Based on the picture above, we can see that if every type of jewelry have false/true value then only append "1" which means every products will be displayed. If both of the conditions are not met, then move on to the else part. When there is one filter, it will only append the "ID LIKE '*%'" (* means the type of jewelry). However, if there are more than one filter, it will append the "OR" and the "ID LIKE '*%'".



```java
//SQL Statement
String sql = stringBuilder.toString();
System.out.println(count);
System.out.println(sql);
ResultSet rs = statement.executeQuery(sql);
//Set the table on Jewelry List as a model
DefaultTableModel model = (DefaultTableModel) tablePane.getModel();
//Import data
while (rs.next()){
    Object[] row = new Object[tablePane.getColumnCount()];
    row[0] = rs.getString( columnLabel: "ID");
    row[1] = typeDecider((String) row[0]);
    row[2] = rs.getString( columnLabel: "Name");
    row[3] = rs.getInt( columnLabel: "Karat");
    row[4] = rs.getFloat( columnLabel: "Weight");
    row[5] = rs.getInt( columnLabel: "Price");
    row[6] = rs.getDate( columnLabel: "Date Added");
    row[7] = rs.getDate( columnLabel: "Date Sold");
    row[8] = checkStock((Integer) rs.getInt( columnLabel: "Quantity"));
    model.addRow(row);
}
```

After that, we convert the StringBuilder to a string since StringBuilder is not a type of string. Therefore, we execute the string and get the specific table model then import it from the database and export it to the specific table. This will be used for jewelry list page.

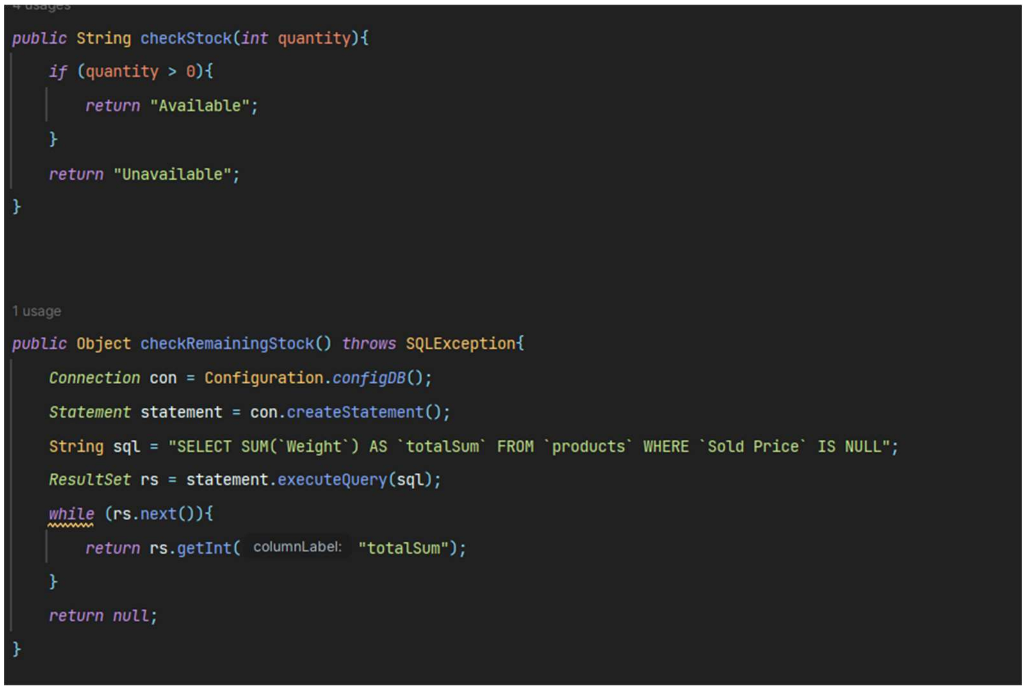
8. calculateExpectedPrice()



```java
public Integer calculateExpectedPrice(int karat, int weight, int profitPercentage) {
    int gold = 1000000;
    double pure = gold * (karat / 24.0); // convert 24 to a double
    int pureWeight = (int) (pure * weight); // convert pure to int
    double profitPercent = profitPercentage / 100.0;
    int profit = (pureWeight*(1+profitPercentage))/10;
    return profit;
}
```

For this method, it is pretty straightforward, I set the value for gold a million. Then I use a bunch of formulas to return the profit results. This will be used for calculating the expected price in the income statement table and price calculator button in the dashboard.

9. checkStock() & checkRemainingStock()

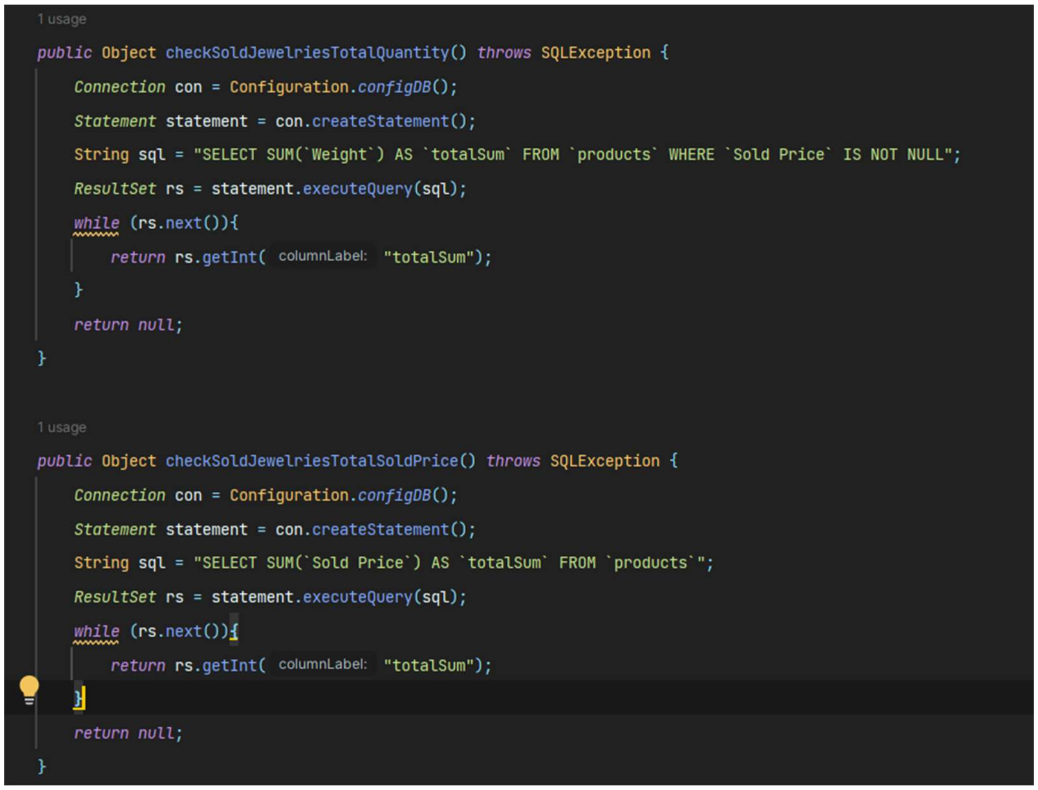


```
public String checkStock(int quantity){
    if (quantity > 0){
        return "Available";
    }
    return "Unavailable";
}



1 usage
public Object checkRemainingStock() throws SQLException{
    Connection con = Configuration.configDB();
    Statement statement = con.createStatement();
    String sql = "SELECT SUM(`Weight`) AS `totalSum` FROM `products` WHERE `Sold Price` IS NULL";
    ResultSet rs = statement.executeQuery(sql);
    while (rs.next()){
        return rs.getInt( columnLabel: "totalSum");
    }
    return null;
}
```

For checkStock(), it only returns a “Available” string if the quantity is more than 0, if not it will be unavailable, this will only be used for the specific tables to tell if the jewelry has been sold or not. This method will be used to determine the availability of a product in a table.

For checkRemainingStock(), it will return the sum of remaining stock from the database by using the query above which means select weight columns value where the ‘sold price’ is empty, from the products. After selecting it, it will sum all of the value and return it. This method will be used for remaining stock box in the dashboard.

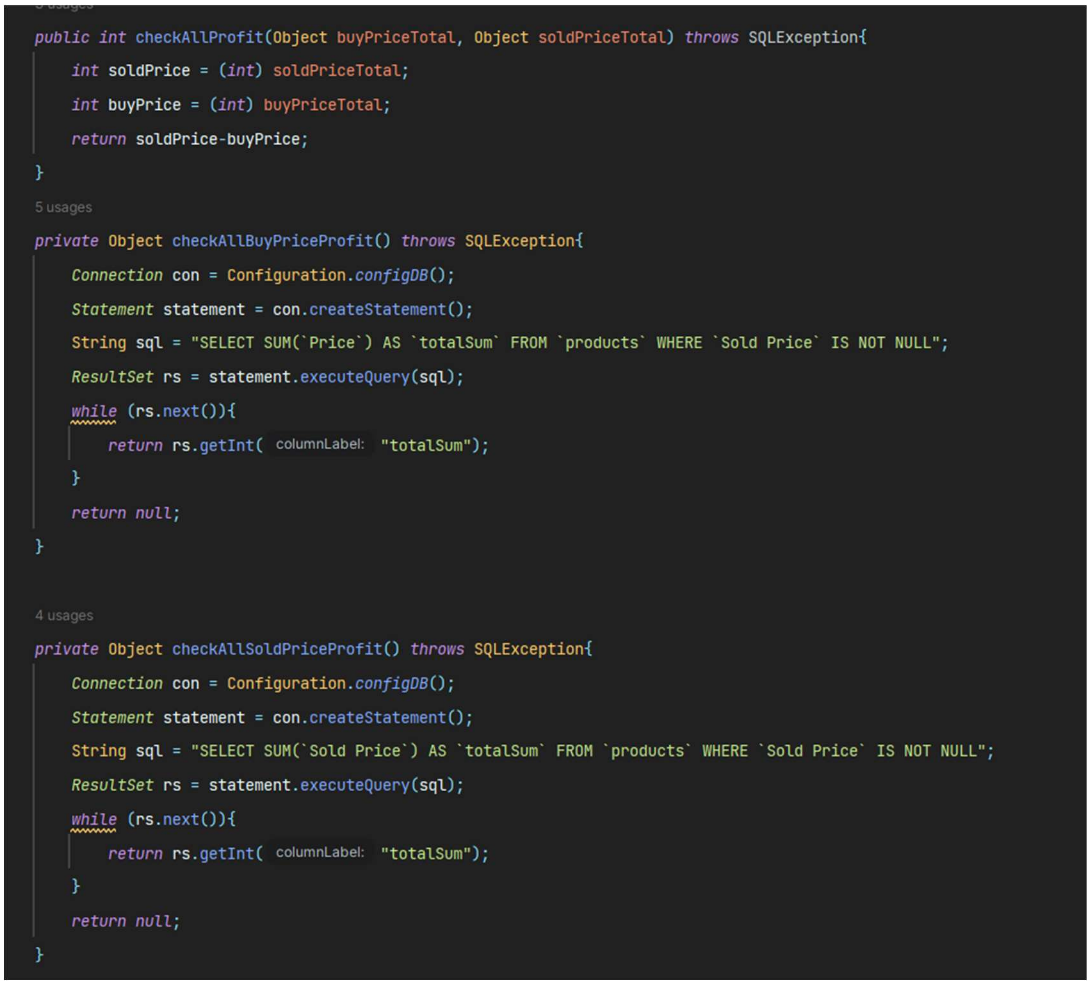
10. checkSoldJewelriesTotalQuantity() & checkSoldJewelriesTotalSoldPrice()



```
1 usage
public Object checkSoldJewelriesTotalQuantity() throws SQLException {
    Connection con = Configuration.configDB();
    Statement statement = con.createStatement();
    String sql = "SELECT SUM(`Weight`) AS `totalSum` FROM `products` WHERE `Sold Price` IS NOT NULL";
    ResultSet rs = statement.executeQuery(sql);
    while (rs.next()){
        return rs.getInt( columnLabel: "totalSum");
    }
    return null;
}

1 usage
public Object checkSoldJewelriesTotalSoldPrice() throws SQLException {
    Connection con = Configuration.configDB();
    Statement statement = con.createStatement();
    String sql = "SELECT SUM(`Sold Price`) AS `totalSum` FROM `products`";
    ResultSet rs = statement.executeQuery(sql);
    while (rs.next()){
        return rs.getInt( columnLabel: "totalSum");
    }
    return null;
}
```

Both of these functions will return an object where they contain the total price and quantity of jewelry that has been sold. For checkSoldJewelriesTotalQuantity(), it will select weight column where the ‘Sold Price’ is not null and return the sum of it. However, for the checkSoldJewelriesTotalSoldPrice() will only sum the ‘Sold Price’ column as a total sum. These methods will be used for sold jewelries box in the dashboard page.

11. checkAllProfit(), checkAllBuyPriceProfit(), & checkAllSoldPriceProfit()



```
public int checkAllProfit(Object buyPriceTotal, Object soldPriceTotal) throws SQLException{
    int soldPrice = (int) soldPriceTotal;
    int buyPrice = (int) buyPriceTotal;
    return soldPrice-buyPrice;
}
5 usages
private Object checkAllBuyPriceProfit() throws SQLException{
    Connection con = Configuration.configDB();
    Statement statement = con.createStatement();
    String sql = "SELECT SUM(`Price`) AS `totalSum` FROM `products` WHERE `Sold Price` IS NOT NULL";
    ResultSet rs = statement.executeQuery(sql);
    while (rs.next()){
        return rs.getInt( columnLabel: "totalSum");
    }
    return null;
}

4 usages
private Object checkAllSoldPriceProfit() throws SQLException{
    Connection con = Configuration.configDB();
    Statement statement = con.createStatement();
    String sql = "SELECT SUM(`Sold Price`) AS `totalSum` FROM `products` WHERE `Sold Price` IS NOT NULL";
    ResultSet rs = statement.executeQuery(sql);
    while (rs.next()){
        return rs.getInt( columnLabel: "totalSum");
    }
    return null;
}
```

For this part, to get all of the profit results, we need to get the sum of buyPriceProfit and soldPriceProfit then we use a function of checkAllProfit() by using this formula "soldPrice-buyPrice". This method will be used for Profit box in dashboard.

C. Dialog

There are many methods that are implemented for this class.

1. calculatorDialog()

   This will open a dialog box where it can calculate the expected price for a type of jewelry. This is also related to the dashboard class where it uses the profit methods to calculate the total profit.

2. soldType()

   This will open a dialog box where it can search a type of jewelry by inputting the code, and determine the sold price for it.

3. inputType()

   This will open a dialog box where it can add a new type of jewelry by inputting the data that are needed for the database, they are name, karat, weight, initial price. For the ID, it will be generated by itself by using another algorithm.

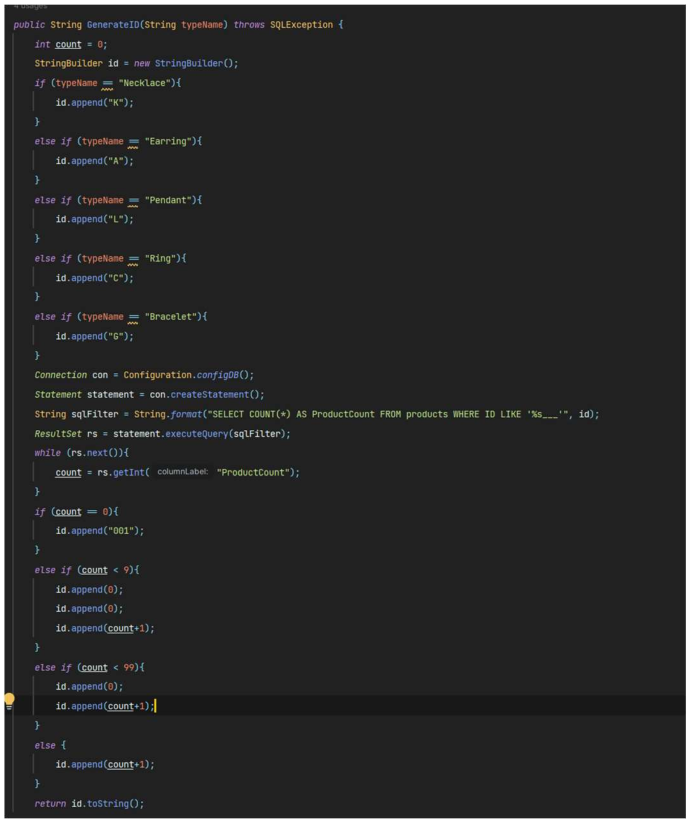
4. searchType()

   This will open a dialog box where it can search a type of jewelry by only inputting the code. Then, by using the SQL query, it will select a product where ID = "<The inputted code>". Then it will pop up from the table.

5. generateID()

   Firstly, I created a string builder so I can append a string easier.

   When given a type of name, it will append the initial name of the jewelry type. Other than that, to get the count number, I use the SQL query to get the count number. To do this, I select any products where the products start with the selected type from the previous initial name then return convert the string builder to string.

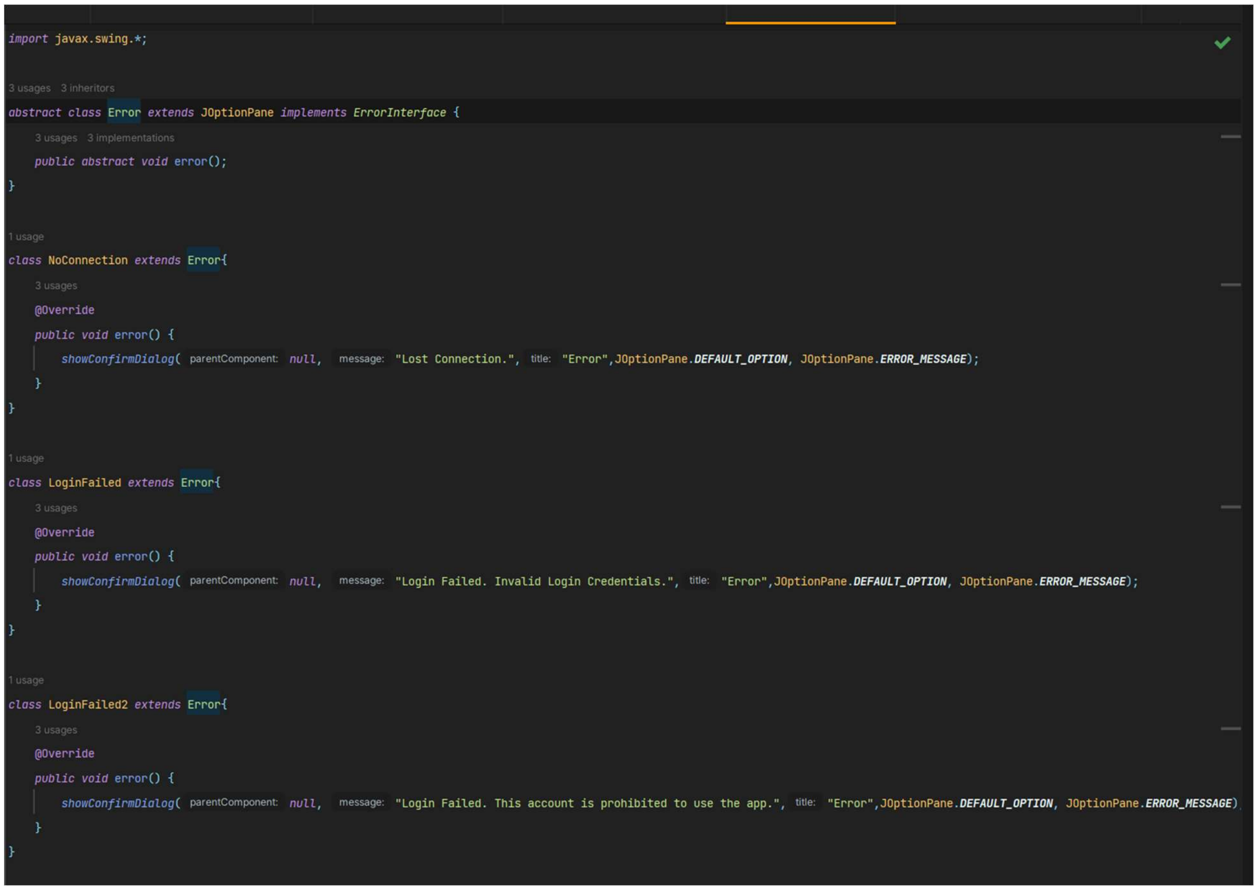


```java
public String GenerateID(String typeName) throws SQLException {
    int count = 0;
    StringBuilder id = new StringBuilder();
    if (typeName == "Necklace"){
        id.append("K");
    }
    else if (typeName == "Earring"){
        id.append("A");
    }
    else if (typeName == "Pendant"){
        id.append("L");
    }
    else if (typeName == "Ring"){
        id.append("C");
    }
    else if (typeName == "Bracelet"){
        id.append("G");
    }
    Connection con = Configuration.configDB();
    Statement statement = con.createStatement();
    String sqlFilter = String.format("SELECT COUNT(*) AS ProductCount FROM products WHERE ID LIKE '%s___'", id);
    ResultSet rs = statement.executeQuery(sqlFilter);
    while (rs.next()){
        count = rs.getInt( columnLabel: "ProductCount");
    }
    if (count == 0){
        id.append("001");
    }
    else if (count < 9){
        id.append(0);
        id.append(0);
        id.append(count+1);
    }
    else if (count < 99){
        id.append(0);
        id.append(count+1);
    }
    else {
        id.append(count+1);
    }
    return id.toString();
```

There are some methods that are already implemented in the dashboard such as soldListIDFilter(), updateTableListWithFilter(), typeDecider(), checkStock(), resetTable(), updateTableList() and calculateExpectedPrice(). This is due to the fact that there is an interface class so both the dashboard and dialog class were implemented by using an interface class.

D. Error

In this error class, I used polymorphism and create a hierarchy of classes. Therefore, there are an additional 4 classes that are included in the error class. First, there is an abstract class of error, and also its method is abstract so it cannot be modified. Then, there is also a class of NoConnection which will show up the no connection error. This also applies to LoginFailed and LoginFailed2 classes, which will show up a confirmation error dialog. To call the error dialog, the developer needs to call the name of the error first and use the error methods.



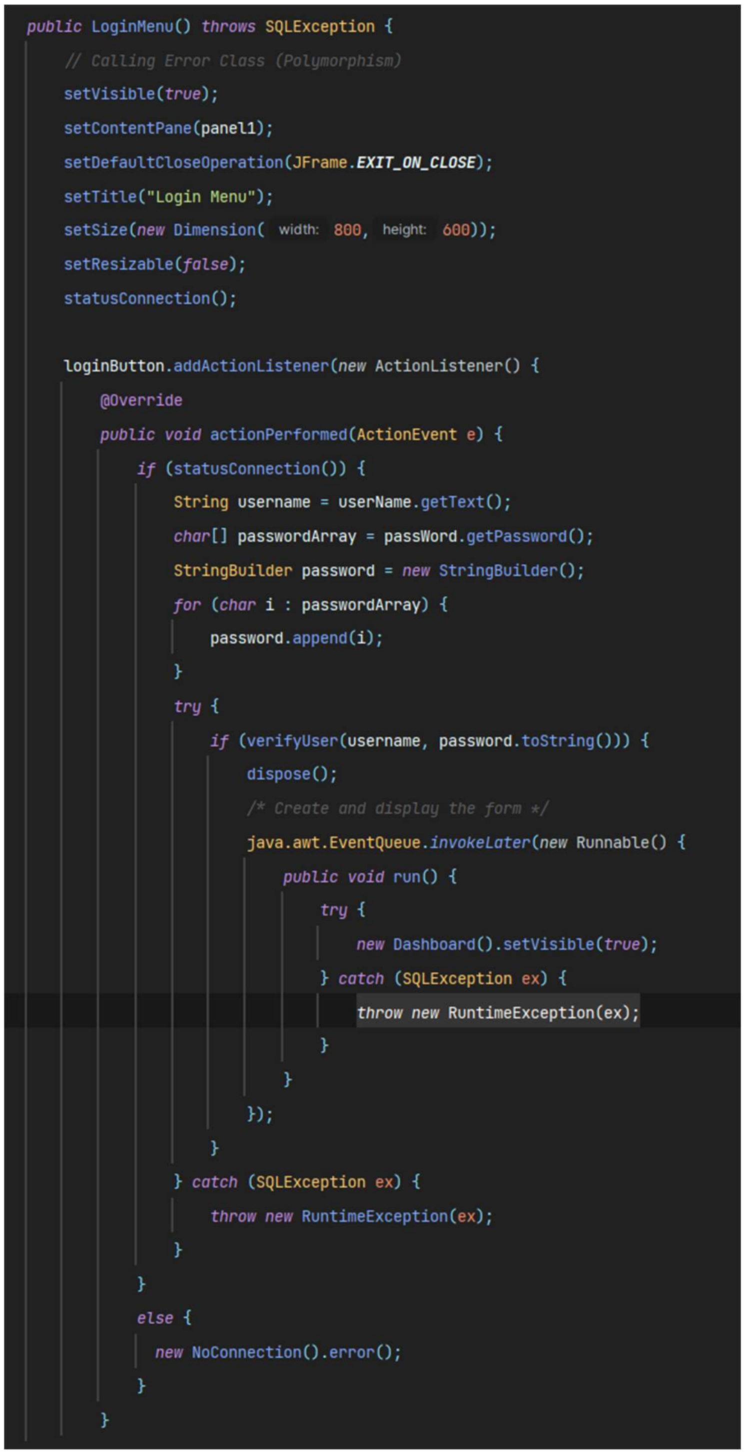
```
import javax.swing.*;

3 usages  3 inheritors
abstract class Error extends JOptionPane implements ErrorInterface {
    3 usages  3 implementations
    public abstract void error();
}

1 usage
class NoConnection extends Error{
    3 usages
    @Override
    public void error() {
        showConfirmDialog( parentComponent: null,  message: "Lost Connection.",  title: "Error",JOptionPane.DEFAULT_OPTION, JOptionPane.ERROR_MESSAGE);
    }
}

1 usage
class LoginFailed extends Error{
    3 usages
    @Override
    public void error() {
        showConfirmDialog( parentComponent: null,  message: "Login Failed. Invalid Login Credentials.",  title: "Error",JOptionPane.DEFAULT_OPTION, JOptionPane.ERROR_MESSAGE);
    }
}

1 usage
class LoginFailed2 extends Error{
    3 usages
    @Override
    public void error() {
        showConfirmDialog( parentComponent: null,  message: "Login Failed. This account is prohibited to use the app.",  title: "Error",JOptionPane.DEFAULT_OPTION, JOptionPane.ERROR_MESSAGE)
    }
}
```

E. Main

For the main class, I only use it to start the login menu page and also set the look and feel (Java Swing theme) to FlatLaf Light which is a custom theme from FlatLaf.

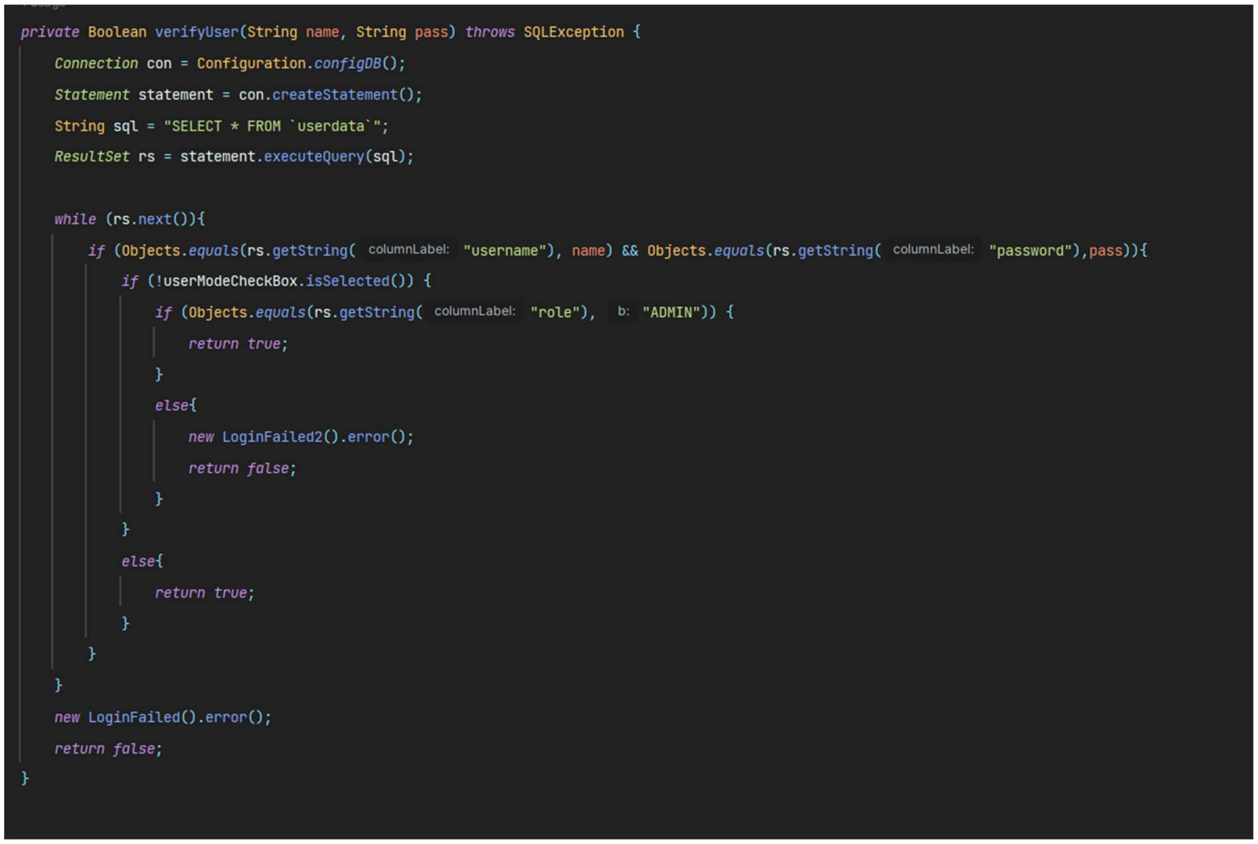
F. LoginMenu

For the login menu, there are not many methods available. However, in this case, I used IntelIJ Idea IDE for designing the login Menu.



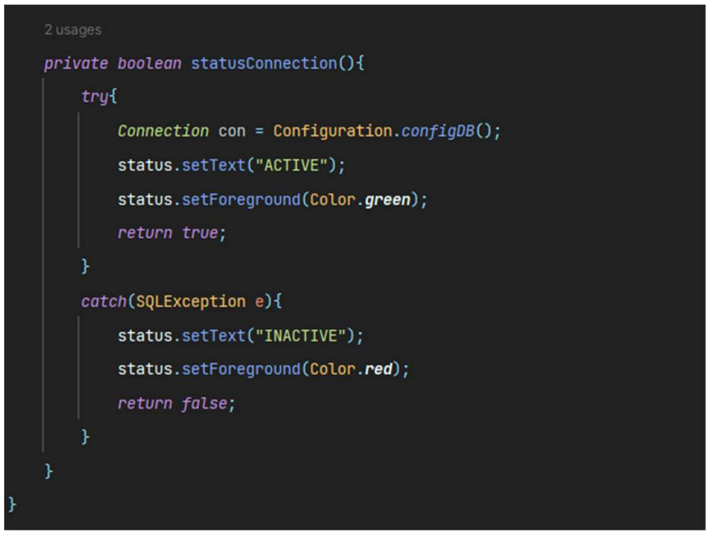
```java
public LoginMenu() throws SQLException {
    // Calling Error Class (Polymorphism)
    setVisible(true);
    setContentPane(panel1);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setTitle("Login Menu");
    setSize(new Dimension( width: 800, height: 600));
    setResizable(false);
    statusConnection();

    loginButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (statusConnection()) {
                String username = userName.getText();
                char[] passwordArray = passWord.getPassword();
                StringBuilder password = new StringBuilder();
                for (char i : passwordArray) {
                    password.append(i);
                }
                try {
                    if (verifyUser(username, password.toString())) {
                        dispose();
                        /* Create and display the form */
                        java.awt.EventQueue.invokeLater(new Runnable() {
                            public void run() {
                                try {
                                    new Dashboard().setVisible(true);
                                } catch (SQLException ex) {
                                    throw new RuntimeException(ex);
                                }
                            }
                        });
                    }
                } catch (SQLException ex) {
                    throw new RuntimeException(ex);
                }
            }
            else {
                new NoConnection().error();
            }
        }
```

When calling the loginMenu(), it will pop up the login menu which consists of a username, password text fields, and also login button. To log in, they need to input a valid username and password otherwise it will call the login failed class as mentioned above. If the SQL is inactive, it will called the no connection class.



```
private Boolean verifyUser(String name, String pass) throws SQLException {
    Connection con = Configuration.configDB();
    Statement statement = con.createStatement();
    String sql = "SELECT * FROM `userdata`";
    ResultSet rs = statement.executeQuery(sql);

    while (rs.next()){
        if (Objects.equals(rs.getString( columnLabel: "username"), name) && Objects.equals(rs.getString( columnLabel: "password"),pass)){
            if (!userModeCheckBox.isSelected()) {
                if (Objects.equals(rs.getString( columnLabel: "role"),  b: "ADMIN")) {
                    return true;
                }
                else{
                    new LoginFailed2().error();
                    return false;
                }
            }
            else{
                return true;
            }
        }
    }
    new LoginFailed().error();
    return false;
}
```

There is also a verifyUser method, so if the username, password, and role respectively correct, it will show the dashboard. Otherwise, it will call the loginFailed error. To verify the user, I use the SQL Query from a userdata database and then execute it to check if the username, password, and role are valid or not.

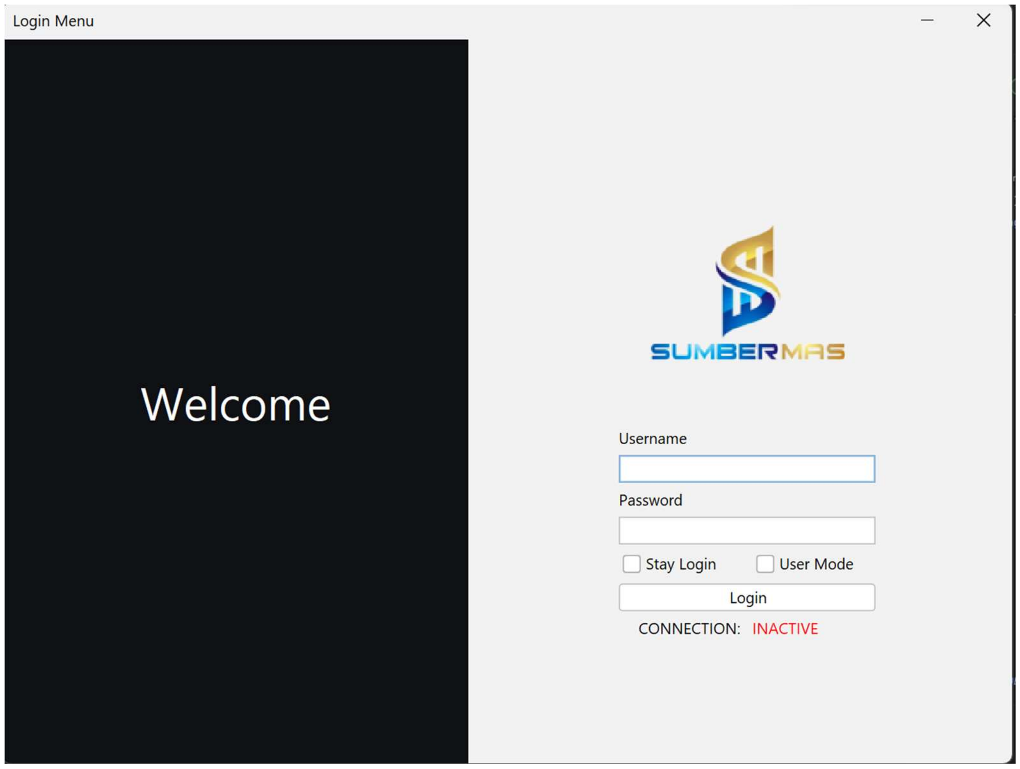


```java
2 usages
private boolean statusConnection(){
    try{
        Connection con = Configuration.configDB();
        status.setText("ACTIVE");
        status.setForeground(Color.green);
        return true;
    }
    catch(SQLException e){
        status.setText("INACTIVE");
        status.setForeground(Color.red);
        return false;
    }
}
}
```

Lastly, statusConnection() method only shows if the SQL Connections is active or inactive and will return a Boolean value.

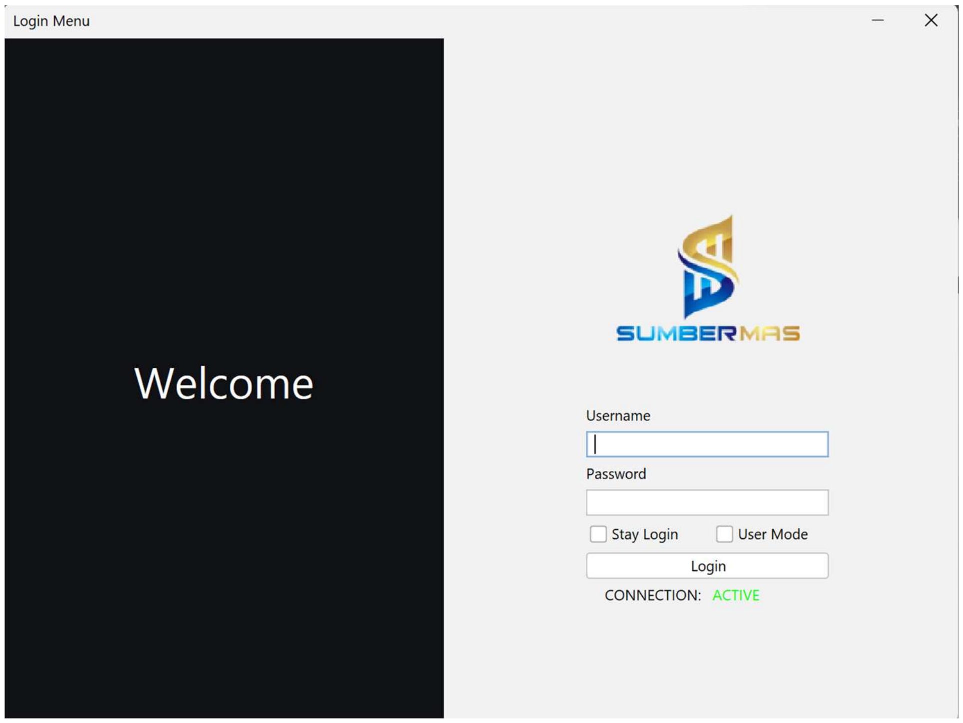
V. Evidence of Working Program

A. Login Menu, inactive SQL server

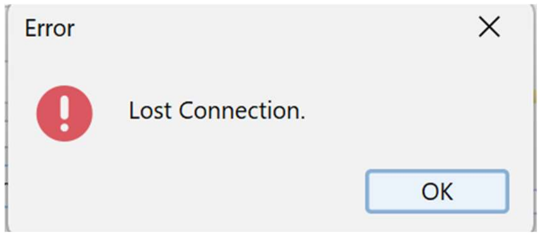


CONNECTION: INACTIVE

B. Login Menu, active SQL Server

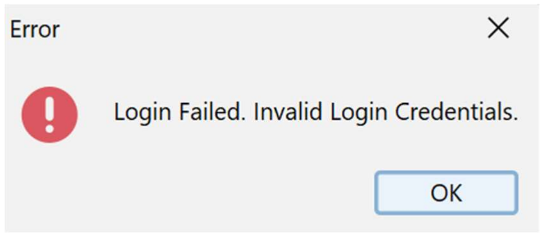


C. Login menu errors


Error

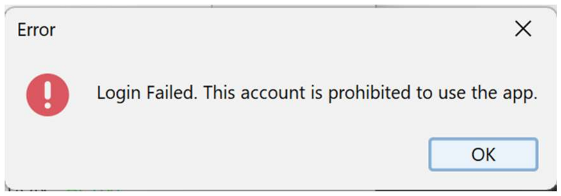
Lost Connection.

OK


Error

Login Failed. Invalid Login Credentials.

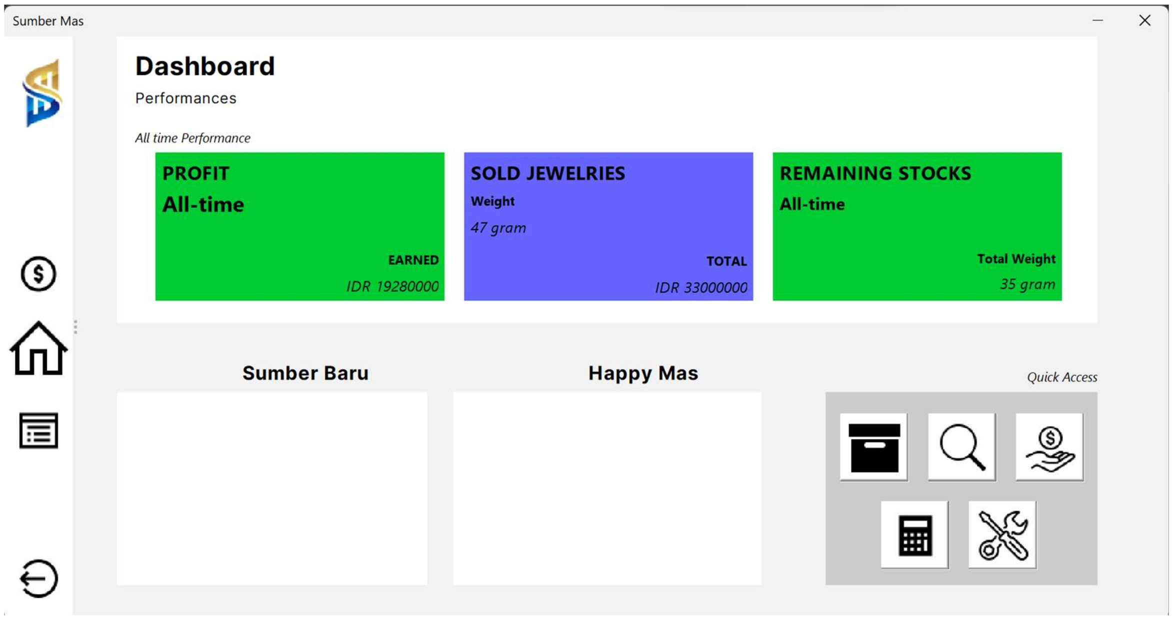
OK


Error

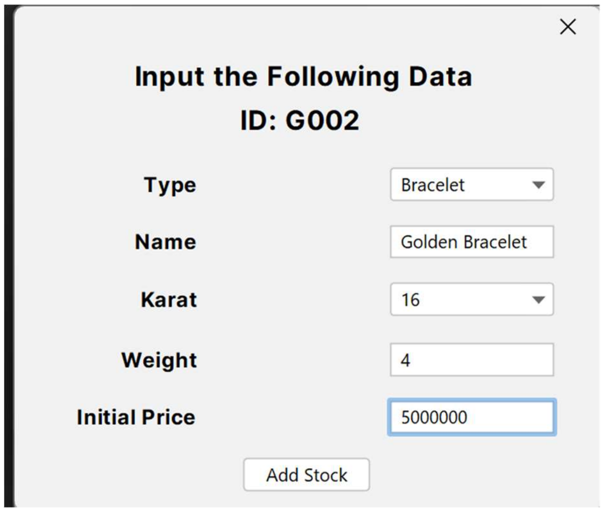
Login Failed. This account is prohibited to use the app.

OK

D. Dashboard

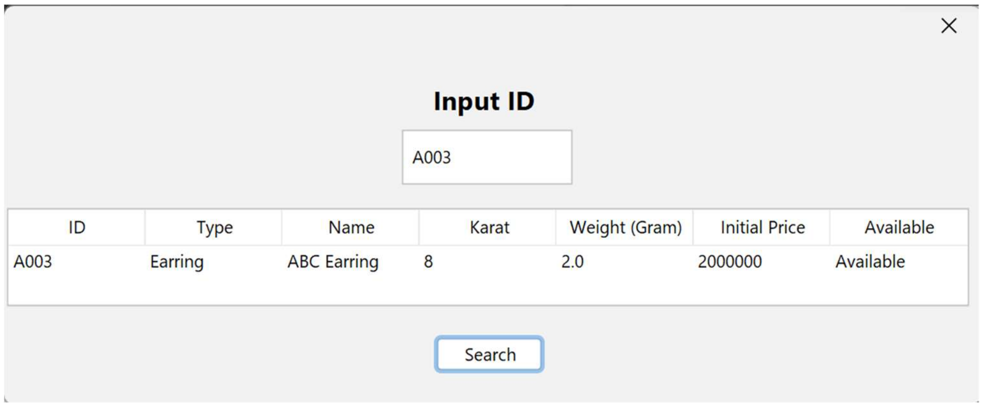


E. Add Stock


Input the Following Data
ID: G002

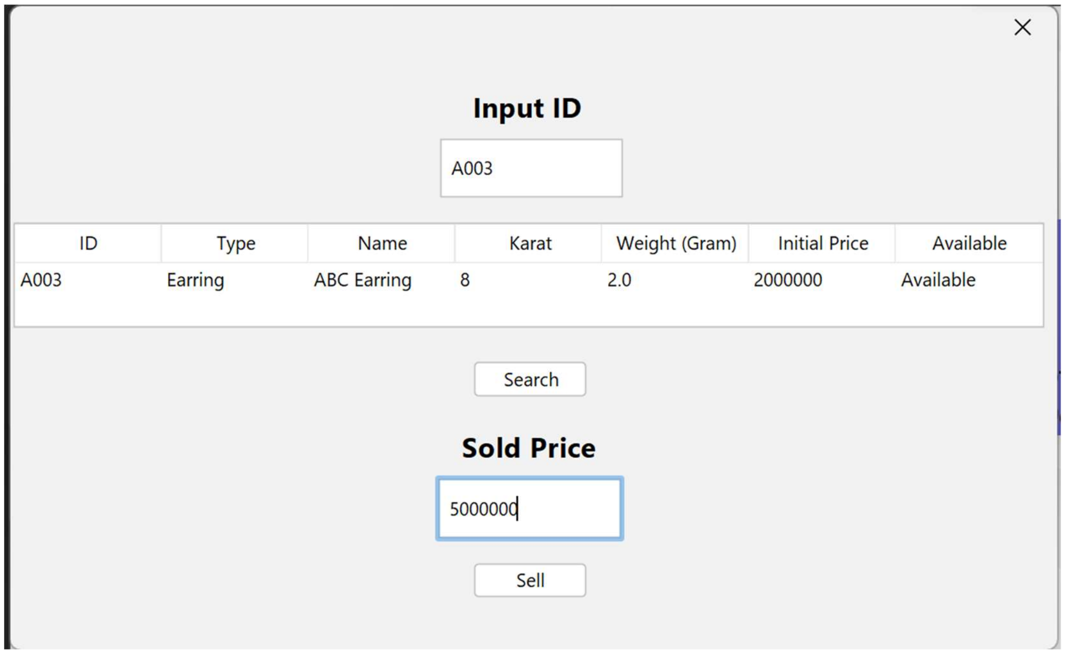
Type	Bracelet
Name	Golden Bracelet
Karat	16
Weight	4
Initial Price	5000000

F. Search


Input ID
A003

ID	Type	Name	Karat	Weight (Gram)	Initial Price	Available
A003	Earring	ABC Earring	8	2.0	2000000	Available

G. Sold Jewelries



Input ID

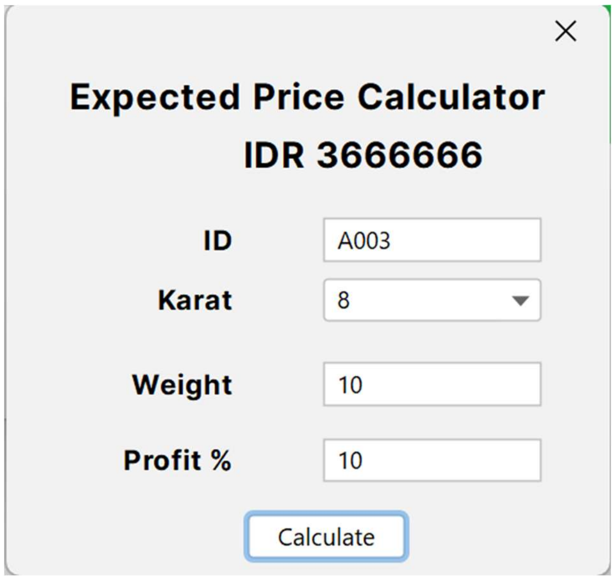
A003

ID	Type	Name	Karat	Weight (Gram)	Initial Price	Available
A003	Earring	ABC Earring	8	2.0	2000000	Available

Sold Price

5000000

H. Expected Price Calculator

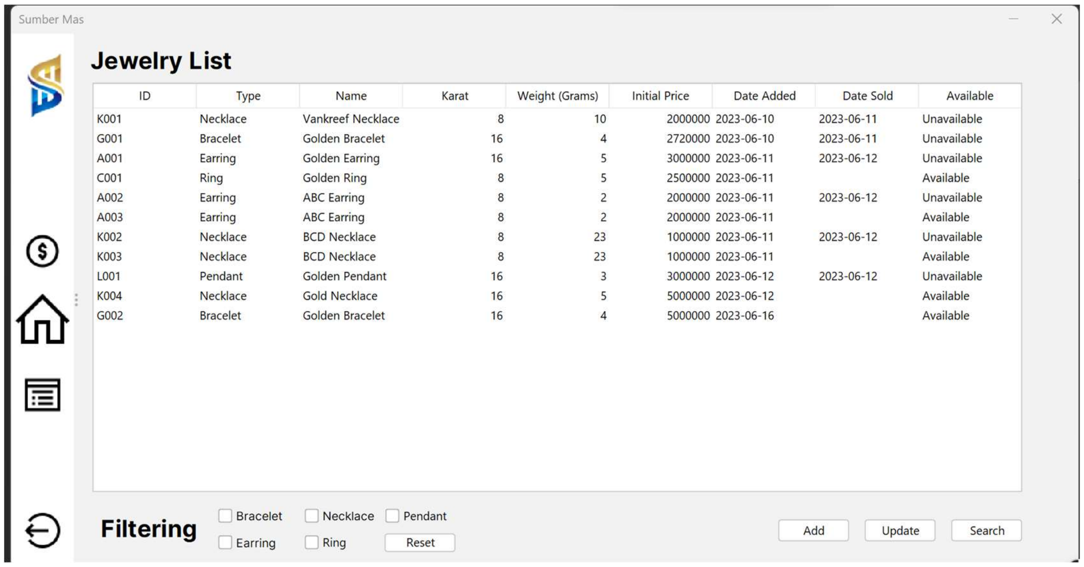


Expected Price Calculator

IDR 3666666

ID	A003
Karat	8
Weight	10
Profit %	10

I. Jewelry List without Filter



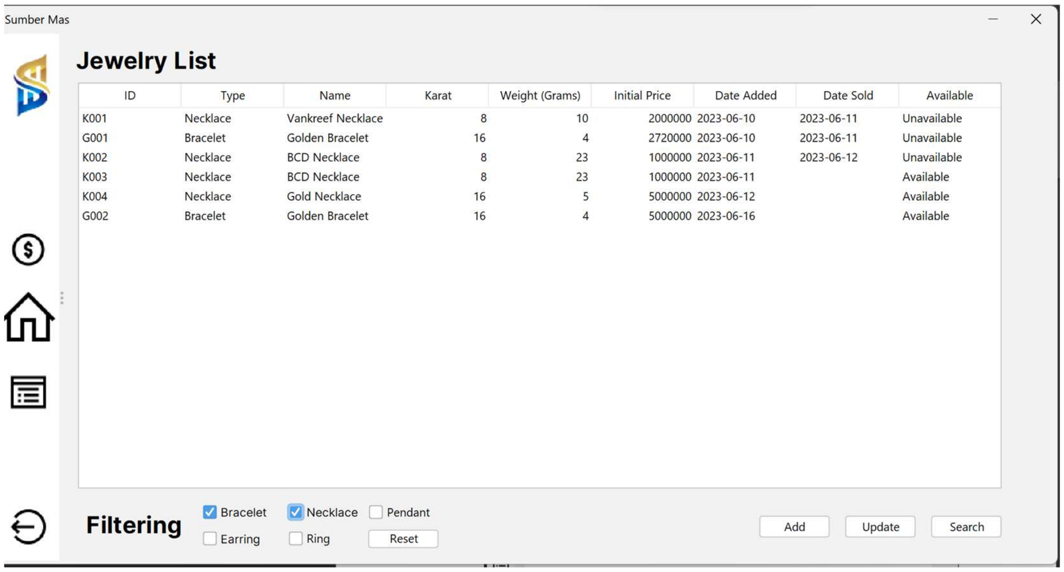
Sumber Mas

Jewelry List

ID	Type	Name	Karat	Weight (Grams)	Initial Price	Date Added	Date Sold	Available
K001	Necklace	Vankreef Necklace	8	10	2000000	2023-06-10	2023-06-11	Unavailable
G001	Bracelet	Golden Bracelet	16	4	2720000	2023-06-10	2023-06-11	Unavailable
A001	Earring	Golden Earring	16	5	3000000	2023-06-11	2023-06-12	Unavailable
C001	Ring	Golden Ring	8	5	2500000	2023-06-11		Available
A002	Earring	ABC Earring	8	2	2000000	2023-06-11	2023-06-12	Unavailable
A003	Earring	ABC Earring	8	2	2000000	2023-06-11		Available
K002	Necklace	BCD Necklace	8	23	1000000	2023-06-11	2023-06-12	Unavailable
K003	Necklace	BCD Necklace	8	23	1000000	2023-06-11		Available
L001	Pendant	Golden Pendant	16	3	3000000	2023-06-12	2023-06-12	Unavailable
K004	Necklace	Gold Necklace	16	5	5000000	2023-06-12		Available
G002	Bracelet	Golden Bracelet	16	4	5000000	2023-06-16		Available

Filtering

J. Jewelry List with Filters



Sumber Mas

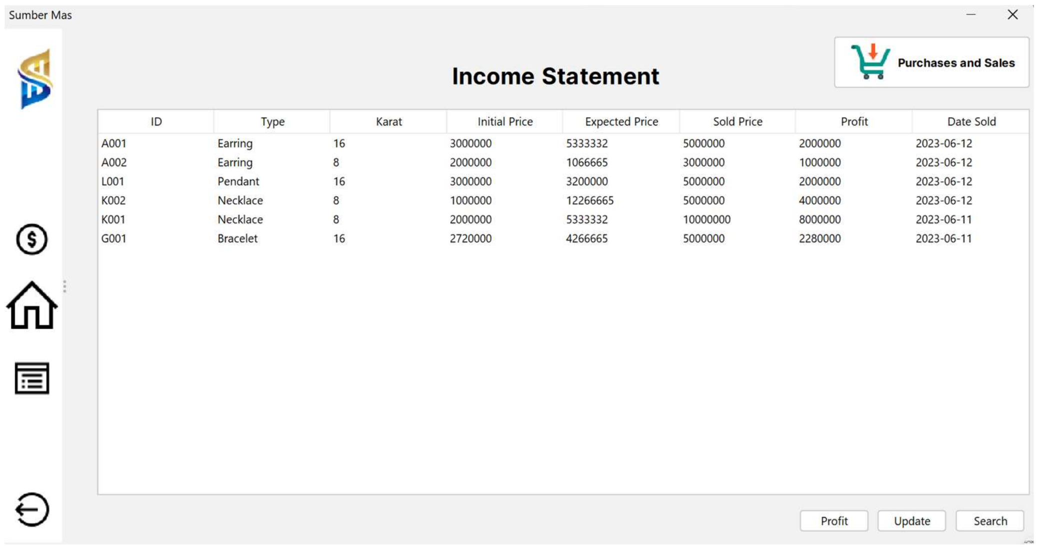
Jewelry List

ID	Type	Name	Karat	Weight (Grams)	Initial Price	Date Added	Date Sold	Available
K001	Necklace	Vankreef Necklace	8	10	2000000	2023-06-10	2023-06-11	Unavailable
G001	Bracelet	Golden Bracelet	16	4	2720000	2023-06-10	2023-06-11	Unavailable
K002	Necklace	BCD Necklace	8	23	1000000	2023-06-11	2023-06-12	Unavailable
K003	Necklace	BCD Necklace	8	23	1000000	2023-06-11		Available
K004	Necklace	Gold Necklace	16	5	5000000	2023-06-12		Available
G002	Bracelet	Golden Bracelet	16	4	5000000	2023-06-16		Available

Filtering

[✓ Bracelet] [✓ Necklace] Pendant
Earring Ring

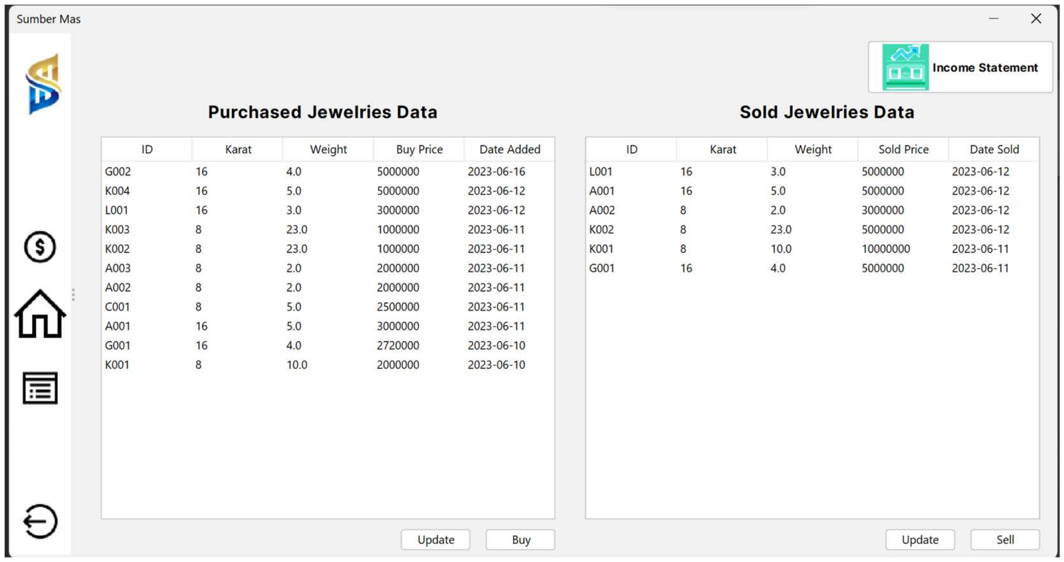
K. Income Statement



Income Statement

ID	Type	Karat	Initial Price	Expected Price	Sold Price	Profit	Date Sold
A001	Earring	16	3000000	5333332	5000000	2000000	2023-06-12
A002	Earring	8	2000000	1066665	3000000	1000000	2023-06-12
L001	Pendant	16	3000000	3200000	5000000	2000000	2023-06-12
K002	Necklace	8	1000000	12266665	5000000	4000000	2023-06-12
K001	Necklace	8	2000000	5333332	10000000	8000000	2023-06-11
G001	Bracelet	16	2720000	4266665	5000000	2280000	2023-06-11

L. Purchased and Sold Jewelry Data



Purchased Jewelries Data

ID	Karat	Weight	Buy Price	Date Added
G002	16	4.0	5000000	2023-06-16
K004	16	5.0	5000000	2023-06-12
L001	16	3.0	3000000	2023-06-12
K003	8	23.0	1000000	2023-06-11
K002	8	23.0	1000000	2023-06-11
A003	8	2.0	2000000	2023-06-11
A002	8	2.0	2000000	2023-06-11
C001	8	5.0	2500000	2023-06-11
A001	16	5.0	3000000	2023-06-11
G001	16	4.0	2720000	2023-06-10
K001	8	10.0	2000000	2023-06-10

Sold Jewelries Data

ID	Karat	Weight	Sold Price	Date Sold
L001	16	3.0	5000000	2023-06-12
A001	16	5.0	5000000	2023-06-12
A002	8	2.0	3000000	2023-06-12
K002	8	23.0	5000000	2023-06-12
K001	8	10.0	10000000	2023-06-11
G001	16	4.0	5000000	2023-06-11