Criterion C - Development
Word Count: 845

The product is a Java program that helps my client manage her home bakery orders and inventory by manipulating data and sending it to and from a database. The goal is to make sure that she can efficiently run her business and manage the daily operations with ease.

Techniques used:

- For loop
- Arrays
- User-defined objects
- Simple and compound selection
- Error handling
- GUI tabs
- GUI popup menus
- GUI buttons
- Use of SQL-Java libraries
- Option pane with user input
- Sending from a database to Java
- Sending data from Java to a database
- Sending data from an HTML form to a database

Structure of my program

- There is a single GUI class that acts as the main class
- The functionality of the program has been broken up into many separate methods
- This works since my program is based on my client interacting with a database she cannot see
- As a result, the program involves her pressing buttons and occasionally manually inputting data (for the inventory)

Through the process of abstraction, I was able to focus on separate methods when needed and also was able to use this to my advantage when it came to the debugging process. In addition, since certain buttons had similar functionality, the modularity of the code I created made it easier and more efficient since I could copy code from one area to another.

The code below is for the button that refreshes the inventory display table for my client. When making this, I was able to copy the code from the method that controls the button which refreshes the orders display table.

```
private void refreshInventoryMouseReleased(java.awt.event.MouseEvent evt) {
    //refreshes the rows of the inventory table to bring the newest updates from the database
    Connection myConn = null;
    Statement myStmt = null;
    ResultSet myRs = null;

    String user = "root";
    String pass = "paggu2004";

    try {
        // 1. Get a connection to database
        myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/orders", user, pass);

        // 2. Create a statement
        myStmt = myConn.createStatement();

        // 3. Execute SQL query
        myRs = myStmt.executeQuery("select * from orders.inventory;");

        // 4. Process the result set
        DefaultTableModel tableModel2 = (DefaultTableModel)inventoryTable.getModel();
        tableModel2.setRowCount(0);
        while (myRs.next()) {

            String quantity = myRs.getString("quantity");
            String ingredientName = myRs.getString("ingredient_name");
            String unit = myRs.getString("unit");

            String[] tableDate = {ingredientName, quantity, unit};
            tableModel2.addRow(tableDate);

        }
            //this warning tells the user that they are running low on any ingredient that is less than 100 in quantity
            for (int i=0;i<tableModel2.getRowCount()-1;i++){
                int x = Integer.parseInt(tableModel2.getValueAt(i,1).toString());
                if (x < 100){
                    Component frame = null;
                    JOptionPane.showMessageDialog(frame, "Running low on ingredients","Ingredient Warning", JOptionPane.WARNING_MESS
```

```
            //this warning tells the user that they are running low on any ingredient that is less than 100 in quantity
            for (int i=0;i<tableModel2.getRowCount()-1;i++){
                int x = Integer.parseInt(tableModel2.getValueAt(i,1).toString());
                if (x < 100){
                    Component frame = null;
                    JOptionPane.showMessageDialog(frame, "Running low on ingredients","Ingredient Warning", JOptionPane.WARNING
                }
            }


        } catch (SQLException e) {
        } finally {
            if (myRs != null) {
                try {
                    myRs.close();
                } catch (SQLException ex) {
                    Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);
                }
            }

            if (myStmt != null) {
                try {
                    myStmt.close();
                } catch (SQLException ex) {
                    Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);
                }
            }

            if (myConn != null) {
                try {
                    myConn.close();
                } catch (SQLException ex) {
                    Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);
                }
            }

        }
    }
```

Below is the code for refreshing the orders display table.

```
    private void refreshButtonMouseReleased(java.awt.event.MouseEvent evt) {
        //refreshes the row to bring the newest orders into the orderList table
        Connection myConn = null;
        Statement myStmt = null;
        ResultSet myRs = null;

        String user = "root";
        String pass = "paggu2004";

        try {
            // 1. Get a connection to database
            myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/orders", user, pass);

            // 2. Create a statement
            myStmt = myConn.createStatement();

            // 3. Execute SQL query
            myRs = myStmt.executeQuery("select * from orders.orders");

            // 4. Process the result set
            DefaultTableModel tableModel = (DefaultTableModel)OrderList.getModel();
            tableModel.setRowCount(0);
            while (myRs.next()) {
                String customerName = myRs.getString("customer_name");
                String phoneNumber = myRs.getString("phone_number");
                String email = myRs.getString("email");
                String orderDate = myRs.getString("order_date");
                String shippingDate = myRs.getString("shipping_date");
                String comments = myRs.getString("comments");
                String items = myRs.getString("food");
                //inserts these variables into a String array to be added to the display table in rows
                String tableDate[] = {customerName, phoneNumber, email, orderDate, shippingDate, comments, items};
                    tableModel.addRow(tableDate);
            }
```

Data structures used:

Many arrays were utilized in my program since data had to be grouped together and stored in display tables. These display tables were used to help make it easy for my client to see her orders and inventory.

This specific array (tableData[]) was used to take in all of the values coming from the MySQL database. It would then be added to the display table.

```
while (myRs.next()) {
    String customerName = myRs.getString("customer_name");
    String phoneNumber = myRs.getString("phone_number");
    String email = myRs.getString("email");
    String orderDate = myRs.getString("order_date");
    String shippingDate = myRs.getString("shipping_date");
    String comments = myRs.getString("comments");
    String items = myRs.getString("food");
    //inserts these variables into a String array to be added to the display table in rows
    String tableDate[] = {customerName, phoneNumber, email, orderDate, shippingDate, comments, items};
        tableModel.addRow(tableDate);
}
```

On the other hand, the array below is utilized as a group of keywords to be searched for within the string of food items that comes from the MySQL database. This is used to see which items need to have ingredients deducted from them after an order is completed.

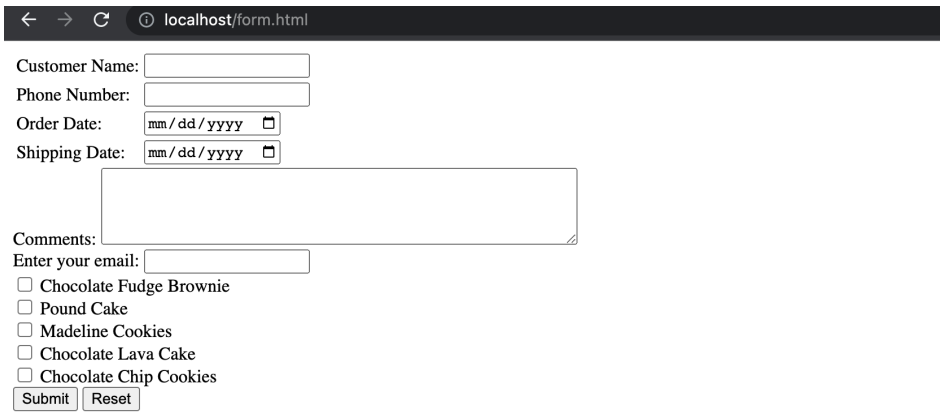
```
//an array with keywords to check the items string with
String[] tokens = {"Brownie", "PoundCake", "Madeline", "ChocoLava", "ChocoChip"};
```

Other algorithms used:

HTML Form

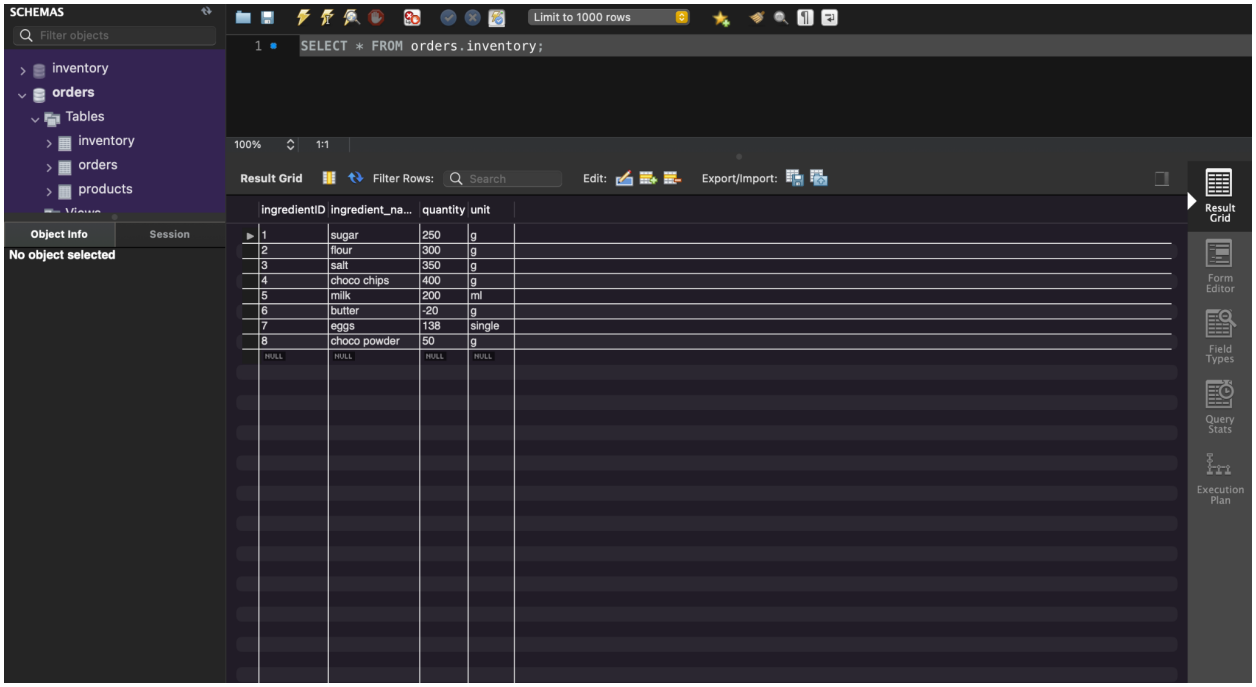
I created a simple HTML (hyper-text markup language) form that could be used by customers to order goods from my client's home bakery. It uses a table to take in certain fields like the customer name and phone number and it uses a checkbox to select which items are being ordered.

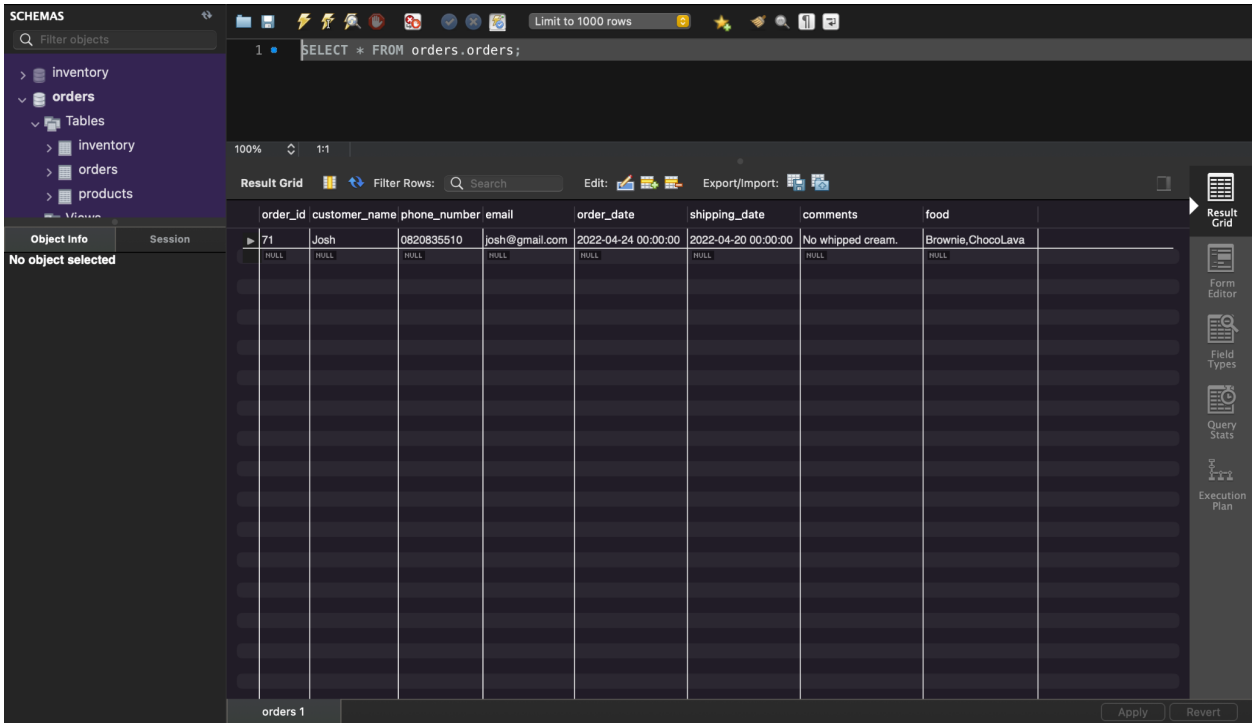
```html
<!DOCTYPE HTML>
<html>
<head>
    <title>Order Form</title>
</head>
<body>
    <form action="database.php" method="POST" enctype="multipart/form-data">
        <table>
            <tr>
                <td>Customer Name: </td>
                <td><input type = "text" id ="Name" name="Name"></td>
            </tr>
            <tr>
                <td>Phone Number: </td>
                <td><input type="text" id ="Phone" name="Phone"></td>
            </tr>
            <tr>
                <td>Order Date: </td>
                <td><input type="Date" id ="OrderDate" name="OrderDate"></td>
            </tr>
            <tr>
                <td>Shipping Date: </td>
                <td><input type="Date" id ="ShipDate" name="ShipDate"></td>
            </tr>
        </table>
        <label for="Comments">Comments:</label>
        <textarea id="Comments" name="Comments" rows="4" cols="50">
        </textarea><br>
        <label for="email">Enter your email:</label>
        <input type="email" id="email" name="email"><br>
        <input type="checkbox" id="Food[]" name="Food[]" value="Brownie">
        <label for="ChocolateFudgeBrownie"> Chocolate Fudge Brownie</label><br>
        <input type="checkbox" id="Food[]" name="Food[]" value="PoundCake">
        <label for="PoundCake"> Pound Cake</label><br>
        <input type="checkbox" id="Food[]" name="Food[]" value="Madeline">
        <label for="Madeline"> Madeline Cookies</label><br>
        <input type="checkbox" id="Food[]" name="Food[]" value="ChocoLava">
        <label for="ChocoLava"> Chocolate Lava Cake</label><br>
        <input type="checkbox" id="Food[]" name="Food[]" value="ChocoChip">
        <label for="ChocoChip"> Chocolate Chip Cookies</label><br>
        <input type="submit" value="Submit">
        <input type="reset">
    </form>
</body>
```



Database

I simply used MySQL workbench to establish my databases for the orders and inventory system. The HTML order form I created sends the data to the orders database.





Order Form to MySQL Database Connection

In order to send the data from the HTML order form to MySQL Workbench, I used phpmyadmin. This code helped to connect the two.

```php
<?php
$dns = '127.0.0.1:3306';
$user = 'root';
$pass = 'paggu2004';
$dbname = 'orders';

$conn = mysqli_connect($dns,$user,$pass,$dbname);

if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}



?>

```

In the code below, I send all of the different values collected by the form to the MySQL Workbench orders database.

```php
<?php
include 'connection.php';
$PhoneNumber = $_POST['Phone'];
$CustomerName = $_POST['Name'];
$Comments = $_POST['Comments'];
$Email = $_POST['email'];
$Food = $_POST['Food'];
$OrderDate = date('Y/m/d');
$ShippingDate = $_POST['ShipDate'];
if(empty($Food))
  {
    echo("You didn't select any items.");
  }
else{
$FoodStr = implode(',',$Food);
}
echo $Email;
$sql = "INSERT INTO orders(customer_name, phone_number, email, order_date, shipping_date, comments, food) VALUES ('".$CustomerName."','".$PhoneNumber."','".$Email."','".$
  OrderDate."','".$ShippingDate."','".$Comments."','".$FoodStr."')";
if ($conn->query($sql) === TRUE) {
  echo "New order created successfully";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();

```

Inserting the orders into the orders display table:

The code below sends the data from MySQL to Java whenever the refresh button is pressed.

- I establish a connection to my database with the object myConn and then create a SQL query with the 'Statement' object myStmt.
- Using the 'ResultSet' object myRs, I execute that query through Java, and from there it runs in MySQL.
- Then it uses a while loop to take all the separate values (customer_name, phone_number, etc.) stored in the orders database and puts it into the aforementioned array and display table.

```
private void refreshButtonMouseReleased(java.awt.event.MouseEvent evt) {
    //refreshes the row to bring the newest orders into the orderList table
    Connection myConn = null;
    Statement myStmt = null;
    ResultSet myRs = null;

    String user = "root";
    String pass = "paggu2004";

    try {
        // 1. Get a connection to database
        myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/orders", user, pass);

        // 2. Create a statement
        myStmt = myConn.createStatement();

        // 3. Execute SQL query
        myRs = myStmt.executeQuery("select * from orders.orders");

        // 4. Process the result set
        DefaultTableModel tableModel = (DefaultTableModel)OrderList.getModel();
        tableModel.setRowCount(0);
        while (myRs.next()) {
            String customerName = myRs.getString("customer_name");
            String phoneNumber = myRs.getString("phone_number");
            String email = myRs.getString("email");
            String orderDate = myRs.getString("order_date");
            String shippingDate = myRs.getString("shipping_date");
            String comments = myRs.getString("comments");
            String items = myRs.getString("food");
            //inserts these variables into a String array to be added to the display table in rows
            String tableDate[] = {customerName, phoneNumber, email, orderDate, shippingDate, comments, items};
                tableModel.addRow(tableDate);
        }
```

Removing an order from the Java display table

The following code removes a row from the orders display table and then proceeds to remove it from the MySQL database. It establishes the connection the same way that other methods do.

```
    private void removeRowMouseReleased(java.awt.event.MouseEvent evt) {
        //deleting a row from the orderList table and removing it from the database
        //get table model
        DefaultTableModel tableModel = (DefaultTableModel)OrderList.getModel();
        //delete selected row
        if(OrderList.getSelectedRowCount()==1){
            //if single row is selected then delete
            String items = OrderList.getValueAt(OrderList.getSelectedRow(), 6).toString();
            tableModel.removeRow(OrderList.getSelectedRow());

            Connection myConn = null;
            Statement myStmt = null;
            ResultSet myRs = null;

            String user = "root";
            String pass = "paggu2004";

        try {
            // 1. Get a connection to database
            myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/orders", user, pass);
        } catch (SQLException ex) {
            Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);
        }

        try {
            // 2. Create a statement
            myStmt = myConn.createStatement();
        } catch (SQLException ex) {
            Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);
        }

        try {
            // 3. Execute SQL query
            myRs = myStmt.executeQuery("select * from orders.orders");
        } catch (SQLException ex) {
            Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);
```

After establishing the connection and running the select SQL query, it finds the order_id of the selected row and proceeds to execute a MySQL update in Java that deletes the row in SQL.

```
        try {
            // 3. Execute SQL query
            myRs = myStmt.executeQuery("select * from orders.orders");
        } catch (SQLException ex) {
            Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);
        }
        int orderID = 0;
        try {
            // 4. Process the result set

            while (myRs.next()) {
                try {
                    orderID = myRs.getInt("order_id");
                } catch (SQLException ex) {
                    Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        } catch (SQLException ex) {
            Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);

        }
            String order_id = Integer.toString(orderID);
            String jdbcUrl = "jdbc:mysql://localhost:3306/orders";
            String sql = "delete from orders where order_id="+order_id;

            try (Connection conn = DriverManager.getConnection(jdbcUrl, user, pass);
            Statement stmt = conn.createStatement();) {

            stmt.executeUpdate(sql);
            System.out.println("Record deleted successfully");
            }
            catch (SQLException ex) {
            Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);
            }
            //an array with keywords to check the items string with
            String[] tokens = {"Brownie", "PoundCake", "Madeline", "ChocoLava", "ChocoChip"};
            //updating the inventory database after finishing an order
            //there is a lot of repetition here as each food item as separate ingredients that it uses
```

Updating the ingredients table

This code updates the ingredients table by deducting certain quantities of ingredients from the MySQL inventory database. That way when the code that is used to refresh the inventory table is run, it shows the remaining amount of each ingredient.

- Each separate if statement refers to what happens if the string items (which is the string of food items ordered) contain one of the 5 tokens in the array.
- It then removes the specific ingredients that are used up by the item that has been ordered by executing a SQL update.

```
String[] tokens = {"Brownie", "PoundCake", "Madeline", "ChocoLava", "ChocoChip"};
//updating the inventory database after finishing an order
//there is a lot of repetition here as each food item as separate ingredients that it uses
if (items.contains(tokens[0])){
    try {
    myStmt.executeQuery("select * from orders.inventory; ");
    } catch (SQLException ex) {
    Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);
    }
    String sql2 = "update orders.inventory set quantity=quantity-100 where ingredient_name='sugar'; ";
    String sql3 = "update orders.inventory set quantity=quantity-50 where ingredient_name='choco powder'; ";
    String sql4 = "update orders.inventory set quantity=quantity-2 where ingredient_name='eggs'; ";
    String sql5 = "update orders.inventory set quantity=quantity-100 where ingredient_name='flour'; ";
    String sql6 = "update orders.inventory set quantity=quantity-100 where ingredient_name='butter';";
    try {
    Connection conn = DriverManager.getConnection(jdbcUrl, user, pass);
    Statement stmt = conn.createStatement();

    stmt.executeUpdate(sql2);
    stmt.executeUpdate(sql3);
    stmt.executeUpdate(sql4);
    stmt.executeUpdate(sql5);
    stmt.executeUpdate(sql6);

    System.out.println("Database updated successfully");
    } catch (SQLException ex) {
    Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);

    }
}
if (items.contains(tokens[1])){
    String sql2 = "update orders.inventory set quantity=quantity-100 where ingredient_name='flour'; ";
    String sql3 = "update orders.inventory set quantity=quantity-2 where ingredient_name='eggs'; ";
    String sql4 = "update orders.inventory set quantity=quantity-100 where ingredient_name='butter'; ";
    String sql5 = "update orders.inventory set quantity=quantity-100 where ingredient_name='sugar';";
    try (Connection conn = DriverManager.getConnection(jdbcUrl, user, pass);
    Statement stmt = conn.createStatement();) {
```

Running low on ingredients message

This simple for loop runs every time the inventory display table is refreshed by my client. It checks every row of the table to see if the amount of ingredients is less than 100. At that point, it opens a JOptionPane which gives a warning to my client.

```
//this warning tells the user that they are running low on any ingredient that is less than 100 in quantity
for (int i=0;i<tableModel2.getRowCount()-1;i++){
    int x = Integer.parseInt(tableModel2.getValueAt(i,1).toString());
    if (x < 100){
        Component frame = null;
        JOptionPane.showMessageDialog(frame, "Running low on ingredients","Ingredient Warning", JOptionPane.WARNING_MESSAGE);
    }
}
```

Updating the quantities in the inventory table

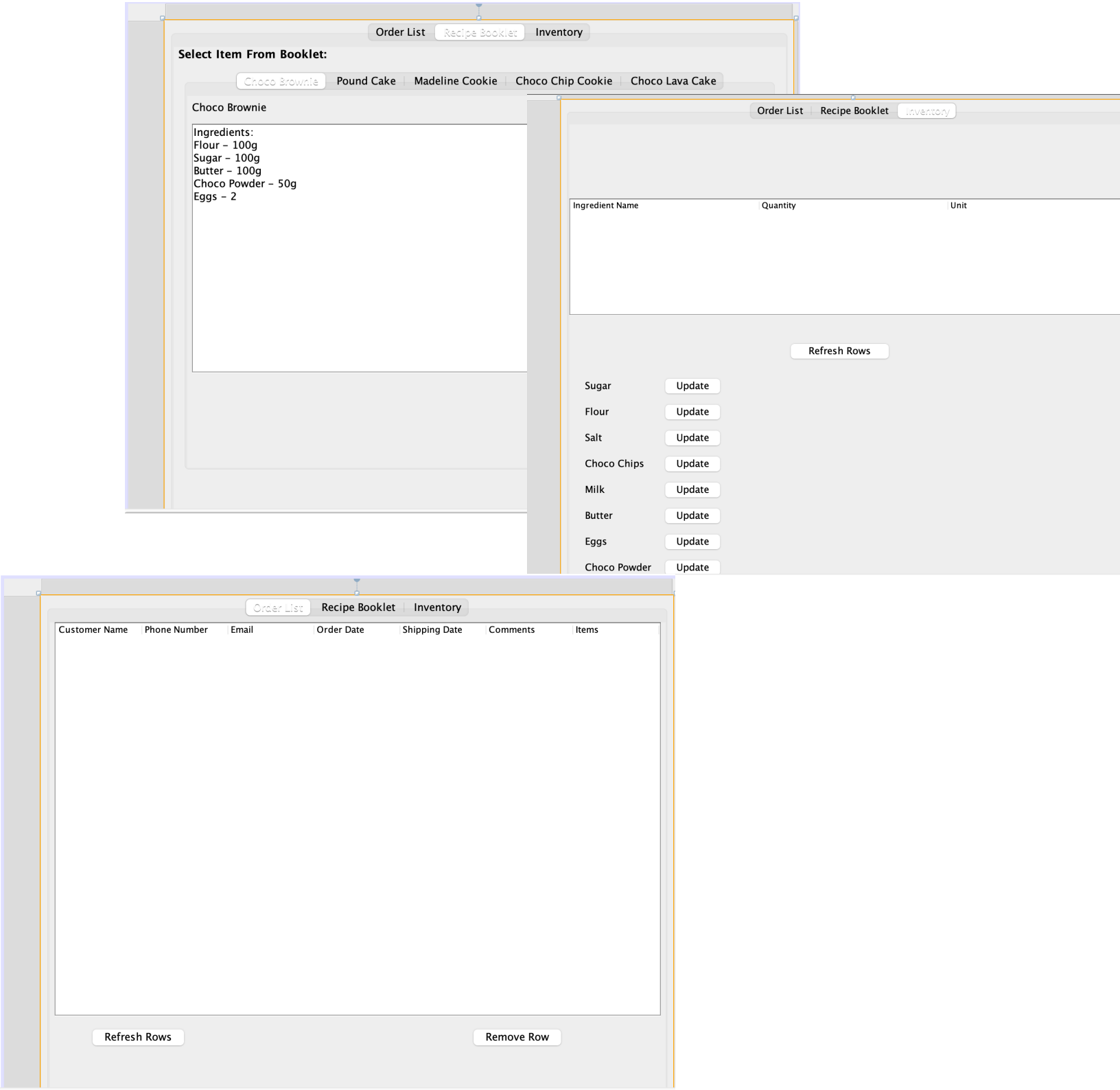
This code allows my client to edit the amount of each ingredient she has.

- First it establishes a connection between the database and the program by creating a 'Connection' object called myConn.
- When my client presses the updater button a JOptionPane pops up allowing her to input a new quantity. This replaces the current quantity with the new one.
- When the table is refreshed, the new amounts appear.
- I have created 8 of these methods, each one represents a different quantity updater button (as there are 8 different ingredients)

```
private void quantityUpdaterMouseReleased(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    String quantityUpdate = JOptionPane.showInputDialog(null,"Enter your quantity: ");

    System.out.println("Quantity is: "+quantityUpdate);
    Connection myConn = null;
    Statement myStmt = null;
    ResultSet myRs = null;

    String user = "root";
    String pass = "paggu2004";

    try {
        // 1. Get a connection to database
        myConn = DriverManager.getConnection("jdbc:mysql://localhost:3306/orders", user, pass);

        // 2. Create a statement
        myStmt = myConn.createStatement();

        // 3. Execute SQL query
        myStmt.executeUpdate("update orders.inventory set quantity="+quantityUpdate+" where ingredient_name='sugar';");



    } catch (SQLException ex) {
        Logger.getLogger(MainGUI.class.getName()).log(Level.SEVERE, null, ex);
    }

}
```

User Interface

For my GUI, I utilized Java Swing components like display tables, tabbed menus, text areas, labels, and many buttons to make my design intuitive. The buttons make my program extremely easy for my client to use as they are quite a simple yet effective UI feature. There are three main tabs that she can easily traverse with the tabbed menu.



Software tools used:

For my program, I utilized NetBeans IDE. It made it very easy to develop my GUI and its visual debugging system made finding errors extremely simple. I also utilized MySQL Workbench to create my database as SQL is an extremely simple language to work with as a result of its lax syntax rule. Finally, I used Sublime text editor to create my HTML form and database connection as a result of the many shortcuts it offers.