Criterion C - Development

Total Word Count: 1064

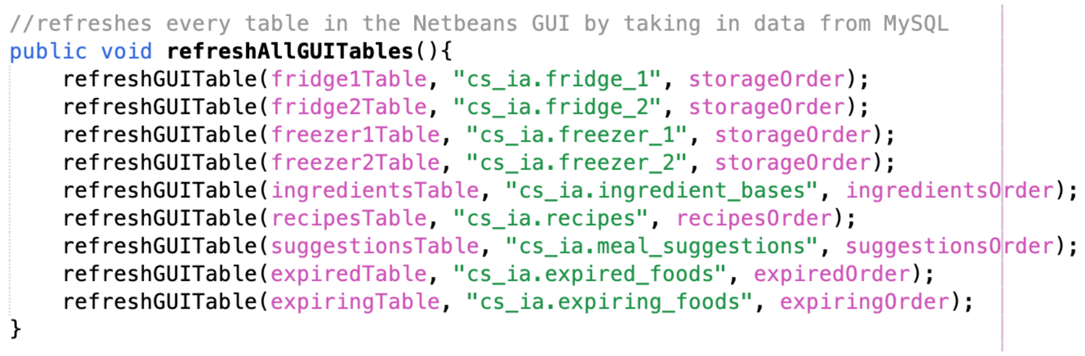
Introduction

I programmed an application that keeps track of the food items stored in my client's house, suggest meals to cook, and notifies her of expiring foods. The Netbeans IDE is used to make a GUI, and MySQL Workbench is used to create a database for the data.

Word Count: 47

Summary of Programming Techniques/Tools

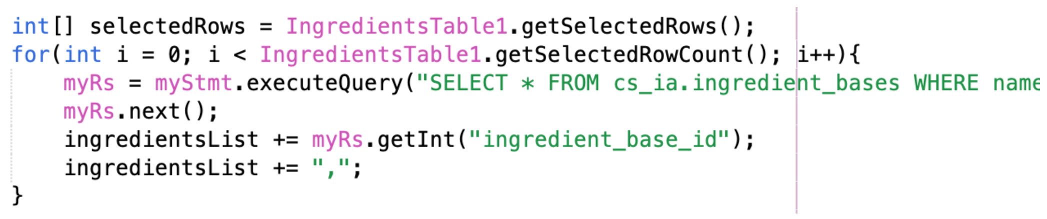
1. For loops
2. While loops
3. Single and compound selection (if/else)
4. Nested for/while loops
5. Concatenation and substring
6. Parameter passing



```
//refreshes every table in the Netbeans GUI by taking in data from MySQL
public void refreshAllGUITables(){
    refreshGUITable(fridge1Table, "cs_ia.fridge_1", storageOrder);
    refreshGUITable(fridge2Table, "cs_ia.fridge_2", storageOrder);
    refreshGUITable(freezer1Table, "cs_ia.freezer_1", storageOrder);
    refreshGUITable(freezer2Table, "cs_ia.freezer_2", storageOrder);
    refreshGUITable(ingredientsTable, "cs_ia.ingredient_bases", ingredientsOrder);
    refreshGUITable(recipesTable, "cs_ia.recipes", recipesOrder);
    refreshGUITable(suggestionsTable, "cs_ia.meal_suggestions", suggestionsOrder);
    refreshGUITable(expiredTable, "cs_ia.expired_foods", expiredOrder);
    refreshGUITable(expiringTable, "cs_ia.expiring_foods", expiringOrder);
}
```

I was working with many tables, and sometimes the same action would need to be performed on each one. Instead of writing the same code ten times, parameters allow the same method to be repurposed. For example, because of the three parameters passed to **refreshGUITables()** this method can correctly refresh the data in any given table.

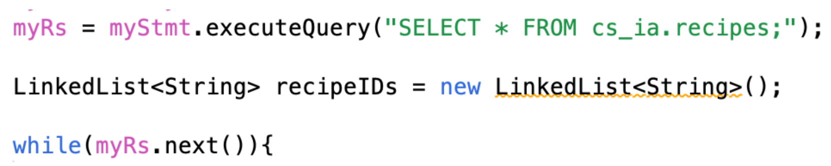
7. Arrays



```
int[] selectedRows = IngredientsTable1.getSelectedRows();
for(int i = 0; i < IngredientsTable1.getSelectedRowCount(); i++){
    myRs = myStmt.executeQuery("SELECT * FROM cs_ia.ingredient_bases WHERE name
    myRs.next();
    ingredientsList += myRs.getInt("ingredient_base_id");
    ingredientsList += ",";
}
```

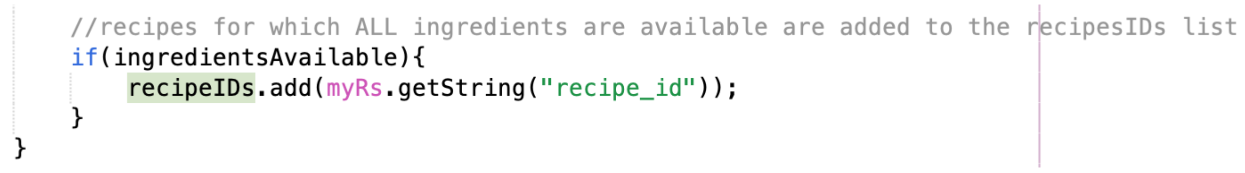
Storing a set of values in an array allowed me to loop through them perform the same action on each element, minimizing repetitive code. Most arrays I declared were used for their purpose and immediately garbage collected, like the one seen above, which stored the indices of the selected rows in a table.

8. Linked lists



```
myRs = myStmt.executeQuery("SELECT * FROM cs_ia.recipes;");

LinkedList<String> recipeIDs = new LinkedList<String>();

while(myRs.next()){
```

[There was more code in between that I removed]

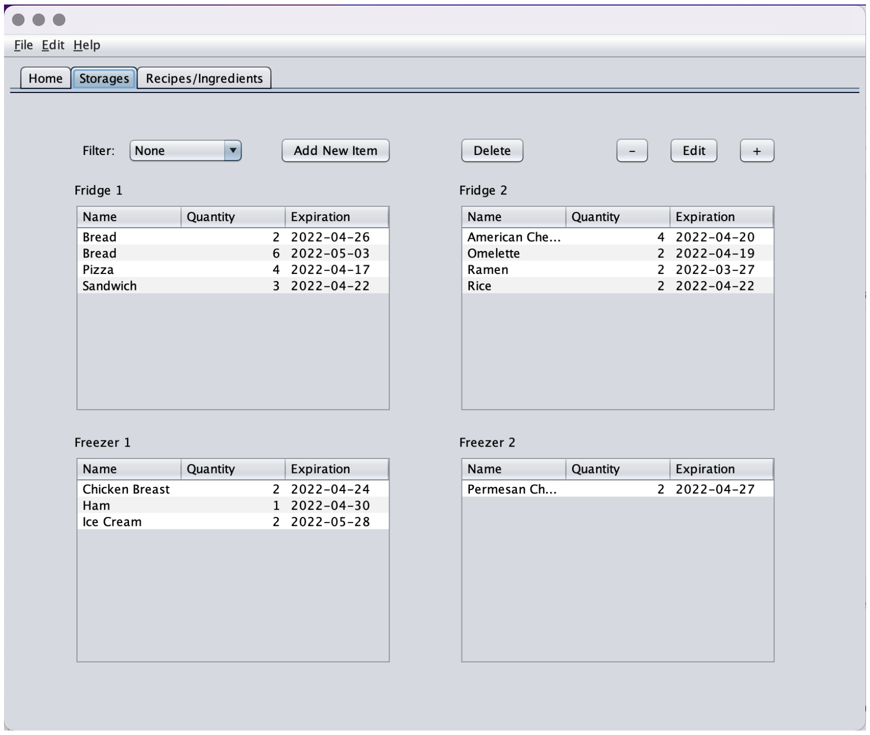


```
    //recipes for which ALL ingredients are available are added to the recipesIDs list
    if(ingredientsAvailable){
        recipeIDs.add(myRs.getString("recipe_id"));
    }
}
```

LinkedLists were used for a similar purpose as arrays, but when the total number of elements wasn't known, since LinkedLists are dynamic. Above, a LinkedList is used to store the values from a result set.

9. GUI interactable elements (JFrame, JTabbedPane, JPanel, JLabel, JButton, JTable, JTextField, JComboBox, JSpinner, JDialog and JOptionPane)

Existing Java Swing tools were used to create a graphical user interface. JPanels within a JTabbedPane separated different parts of the program into tabs, JTables displayed data, JButtons added, deleted, and modified the data, and JTextFields/JComboBoxes/JSpinners were used to input various types of information. Popup menus were also generated to notify the user of errors or allow for extra user input that didn't clutter the original tab.



Filter: None

Fridge 1

Name	Quantity	Expiration
Bread	2	2022-04-26
Bread	6	2022-05-03
Pizza	4	2022-04-17
Sandwich	3	2022-04-22

Fridge 2

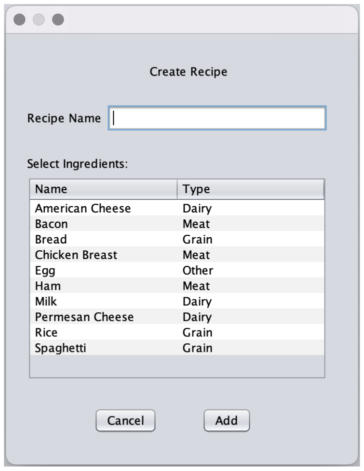
Name	Quantity	Expiration
American Che...	4	2022-04-20
Omelette	2	2022-04-19
Ramen	2	2022-03-27
Rice	2	2022-04-22

Freezer 1

Name	Quantity	Expiration
Chicken Breast	2	2022-04-24
Ham	1	2022-04-30
Ice Cream	2	2022-05-28

Freezer 2

Name	Quantity	Expiration
Permesan Ch...	2	2022-04-27



Create Recipe

Recipe Name

Select Ingredients:

Name	Type
American Cheese	Dairy
Bacon	Meat
Bread	Grain
Chicken Breast	Meat
Egg	Other
Ham	Meat
Milk	Dairy
Permesan Cheese	Dairy
Rice	Grain
Spaghetti	Grain

10. Event listeners

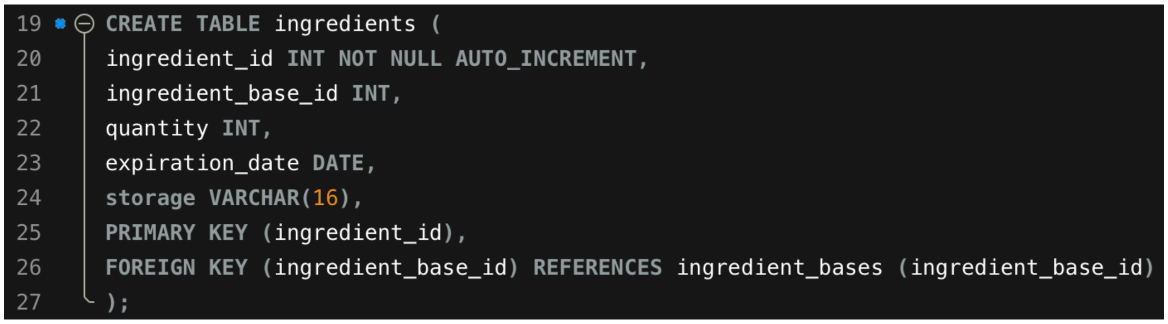
Existing event listeners like **mouseReleased** and **itemStateChange** were used in Netbeans to run code upon user interaction with the GUI.

11. Nested classes

```
//nested class TableHeaderMouseListener is used to check for clicks on table headers
//code adapted from https://www.codejava.net/java-se/swing/how-to-handle-mouse-clicking-event-on-jtable-column-header
class TableHeaderMouseListener extends MouseAdapter {
    private JTable GUITable;

    private TableHeaderMouseListener(JTable GUITable){
        this.GUITable = GUITable;
    }

    public void mouseClicked(MouseEvent evt) {
        java.awt.Point point = evt.getPoint();
        int column = GUITable.columnAtPoint(point);

        String[] ingredientsOrders = {"name", "type"};
        String[] storageOrders = {"name", "quantity", "expiration_date"};

        //ordering parameters are re-assigned for ingredient/storage tables upon header click
        //the values in the given table(s) will be ordered by the column clicked
        if(GUITable == ingredientsTable || GUITable == ingredientsTable1 || GUITable == ingredientsTable2){
            ingredientsOrder = ingredientsOrders[column];
        }else{
            storageOrder = storageOrders[column];
        }

        //updates the GUI tables with the new ordering parameters
        refreshAllGUITables();
    }
}
```

My code used one nested class, which was the **TableHeaderMouseListener** class. This class extends **MouseAdapter** and is used to create MouseEvents that trigger upon clicking the header of a table[1]. This class was created inside of **MainGUI** instead of as a separate class to bypass an error that arose from referencing a non-static method from a static context.

12. Create an SQL database[2]
13. Create tables in SQL
    a. Primary and foreign keys
    b. NOT NULL AUTO_INCREMENT keyword



```sql
CREATE TABLE ingredients (
ingredient_id INT NOT NULL AUTO_INCREMENT,
ingredient_base_id INT,
quantity INT,
expiration_date DATE,
storage VARCHAR(16),
PRIMARY KEY (ingredient_id),
FOREIGN KEY (ingredient_base_id) REFERENCES ingredient_bases (ingredient_base_id)
);
```

Tables were created in MySQL Workbench using the CREATE TABLE command. Primary keys were constrained with NOT NULL AUTO_INCREMENT so that each row would always contain a unique PK. FKs were used to access values from other tables, like how each ingredient takes on the name and type of its corresponding ingredient_base.

Some tables were created as "intermediary" tables, used to collate values from other tables before being ordered by a certain parameter and used in Netbeans. They served the purpose of views, except that views don't support multiple JOIN statements (bullet #25).

14. Java Database Connectivity (JDBC)[3]

A connection was established between Netbeans and MySQL to allow for queries to be executed as a result of some user interaction.

[1] The code in this class was adapted from an online source. Citation: Minh, N. H. (2019, July 4). *How to handle mouse clicking event on JTable column header*. CodeJava. https://www.codejava.net/java-se/swing/how-to-handle-mouse-clicking-event-on-jtable-column-header

[2] I learned to use MySQL workbench and all the SQL keywords mentioned in this document through a video. Citation: Programming with Mosh. (2019). MySQL Tutorial for Beginners [Full Course] [YouTube Video]. In *YouTube*. https://www.youtube.com/watch?v=7S_tz1z_5bA

[3] I learned to use a JDBC, Statements, and ResultSets through a playlist of videos. Citation: luv2code. (2014, March 31). *Java JDBC Tutorial - Part 0: Overview*. Www.youtube.com. https://www.youtube.com/watch?v=8-iQDUl10vM&list=PLEAQNNR8IlB4R7NfqBY1frapYo97L6fOQ

15. Try/catch
16. Statement

```java
for(int i = 0; i < recipeIDs.size(); i++){
    myStmt.executeUpdate("INSERT INTO cs_ia.meal_suggestions" +
                        " (name, ingredients_list) " +
                        "SELECT name, ingredients_list " +
                        "FROM recipes " +
                        "WHERE recipe_id = '" + recipeIDs.get(i) + "';");
}
```

Statements were used to run commands like INSERT and UPDATE, changing the database based on user inputs.

17. ResultSet

```java
while(myRs.next()){
    String name = myRs.getString("name");
    String quantity = myRs.getString("quantity");
    String daysLeft = myRs.getString("expiration_date");
    String[] tableData = {name, quantity, daysLeft};
    tableModel.addRow(tableData);
}
```

Result sets were used in conjunction with statements to SELECT data from the database. This was mainly used to populate JTables in the GUI.

18. Select clause
19. Insert clause
20. Update clause
21. Where operator
22. Delete command
23. Order by operator
24. Inner join

```java
myStmt.executeUpdate("INSERT INTO " + storage +
                    " (name, quantity, expiration_date) " +
                    "SELECT name, quantity, expiration_date " +
                    "FROM ingredients " +
                    "INNER JOIN ingredient_bases " +
                    "ON ingredients.ingredient_base_id = ingredient_bases.ingredient_base_id " +
                    "WHERE storage = '" + storage + "';");
```

The INNER JOIN keyword was used to insert data from multiple tables into the aforementioned “intermediary” tables. Above, a JOIN is used to select the quantity and expiration_date of an ingredient, as well as the name of the ingredient_base with the same ingredient_base_id key as the ingredient.

Word Count: 494 (not including bulleted lists)

Structure of Program

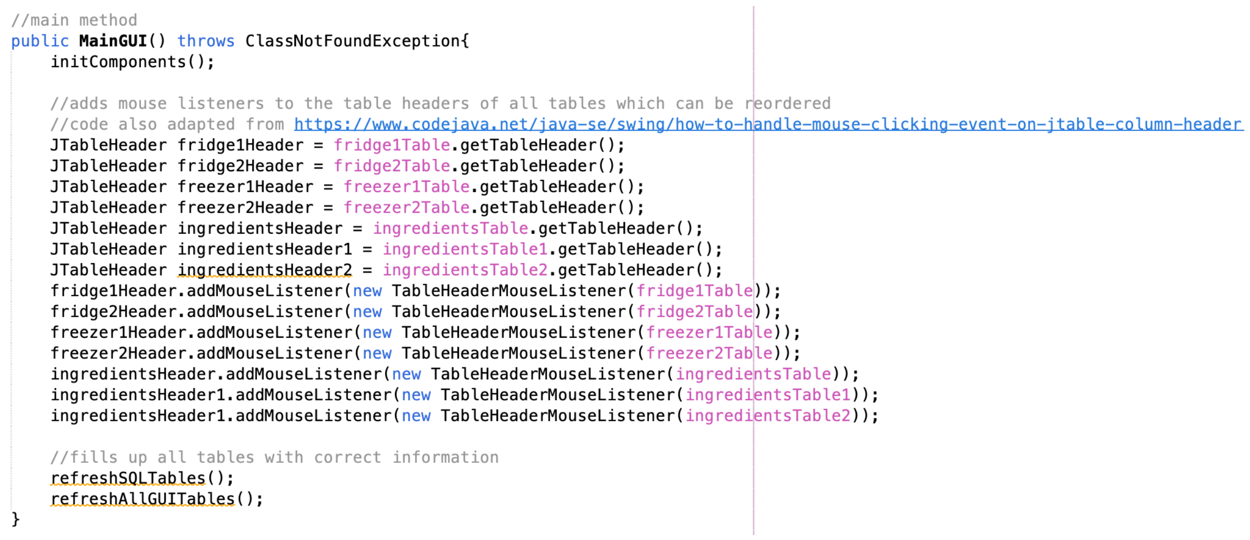
The GUI is split into three tabs: “Home,” where the user is displayed with important information, “Storages,” where the user sees and can interact with all food items stored in the house, and “Recipes/Ingredients,” where the user can create and edit recipes and base ingredients.

The program itself doesn’t have much of a structure, as it’s highly dependent on which of the many GUI elements in each tab the user decides to interact with. After the main method is called, user interaction with GUI elements dictates the flow of the program.

Word Count: 92

Main Algorithms

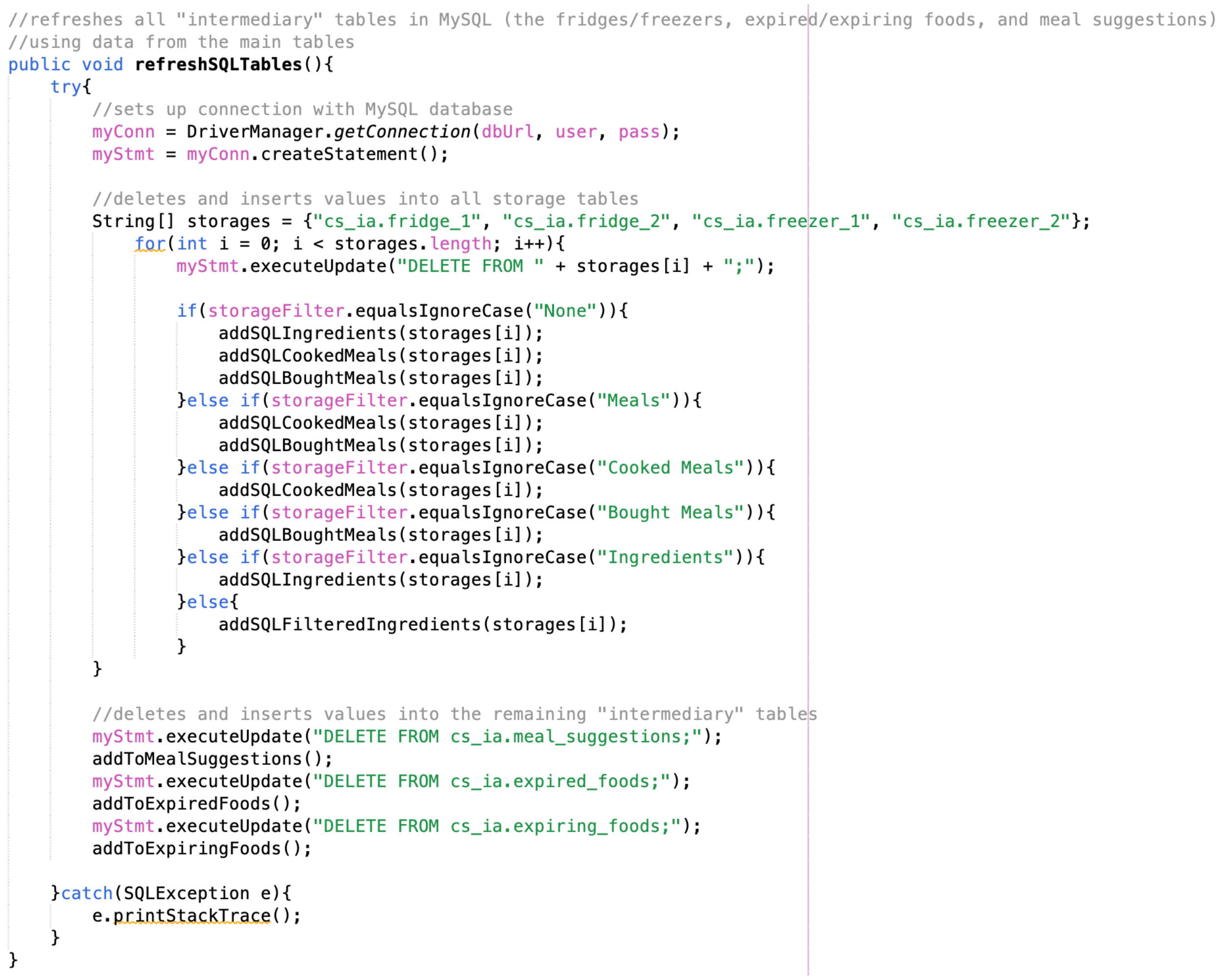
1. Main



```
//main method
public MainGUI() throws ClassNotFoundException{
    initComponents();

    //adds mouse listeners to the table headers of all tables which can be reordered
    //code also adapted from https://www.codejava.net/java-se/swing/how-to-handle-mouse-clicking-event-on-jtable-column-header
    JTableHeader fridge1Header = fridge1Table.getTableHeader();
    JTableHeader fridge2Header = fridge2Table.getTableHeader();
    JTableHeader freezer1Header = freezer1Table.getTableHeader();
    JTableHeader freezer2Header = freezer2Table.getTableHeader();
    JTableHeader ingredientsHeader = ingredientsTable.getTableHeader();
    JTableHeader ingredientsHeader1 = ingredientsTable1.getTableHeader();
    JTableHeader ingredientsHeader2 = ingredientsTable2.getTableHeader();
    fridge1Header.addMouseListener(new TableHeaderMouseListener(fridge1Table));
    fridge2Header.addMouseListener(new TableHeaderMouseListener(fridge2Table));
    freezer1Header.addMouseListener(new TableHeaderMouseListener(freezer1Table));
    freezer2Header.addMouseListener(new TableHeaderMouseListener(freezer2Table));
    ingredientsHeader.addMouseListener(new TableHeaderMouseListener(ingredientsTable));
    ingredientsHeader1.addMouseListener(new TableHeaderMouseListener(ingredientsTable1));
    ingredientsHeader1.addMouseListener(new TableHeaderMouseListener(ingredientsTable2));

    //fills up all tables with correct information
    refreshSQLTables();
    refreshAllGUITables();
}
```

The main method uses the **TableHeaderMouseListener** nested class to add mouse listeners to the headers of every sortable JTable. It then calls on two other methods to refresh the data in MySQL “intermediary” tables and all GUI tables.

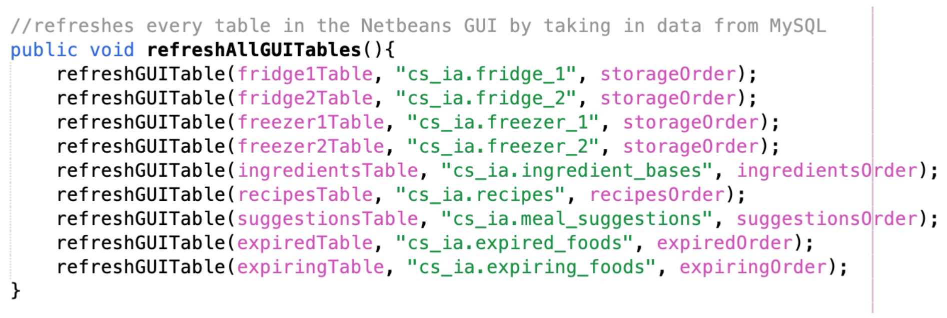
2. Refreshing SQL tables



```java
//refreshes all "intermediary" tables in MySQL (the fridges/freezers, expired/expiring foods, and meal suggestions)
//using data from the main tables
public void refreshSQLTables(){
    try{
        //sets up connection with MySQL database
        myConn = DriverManager.getConnection(dbUrl, user, pass);
        myStmt = myConn.createStatement();

        //deletes and inserts values into all storage tables
        String[] storages = {"cs_ia.fridge_1", "cs_ia.fridge_2", "cs_ia.freezer_1", "cs_ia.freezer_2"};
            for(int i = 0; i < storages.length; i++){
                myStmt.executeUpdate("DELETE FROM " + storages[i] + ";");

                if(storageFilter.equalsIgnoreCase("None")){
                    addSQLIngredients(storages[i]);
                    addSQLCookedMeals(storages[i]);
                    addSQLBoughtMeals(storages[i]);
                }else if(storageFilter.equalsIgnoreCase("Meals")){
                    addSQLCookedMeals(storages[i]);
                    addSQLBoughtMeals(storages[i]);
                }else if(storageFilter.equalsIgnoreCase("Cooked Meals")){
                    addSQLCookedMeals(storages[i]);
                }else if(storageFilter.equalsIgnoreCase("Bought Meals")){
                    addSQLBoughtMeals(storages[i]);
                }else if(storageFilter.equalsIgnoreCase("Ingredients")){
                    addSQLIngredients(storages[i]);
                }else{
                    addSQLFilteredIngredients(storages[i]);
                }
        }

        //deletes and inserts values into the remaining "intermediary" tables
        myStmt.executeUpdate("DELETE FROM cs_ia.meal_suggestions;");
        addToMealSuggestions();
        myStmt.executeUpdate("DELETE FROM cs_ia.expired_foods;");
        addToExpiredFoods();
        myStmt.executeUpdate("DELETE FROM cs_ia.expiring_foods;");
        addToExpiringFoods();

    }catch(SQLException e){
        e.printStackTrace();
    }
}
```

refreshSQLTables() uses Statements to delete data from every “intermediary” table in MySQL, then adds values back in to each table by calling other methods. These methods were separated to not clutter **refreshSQLTables()**.

3. Refreshing JTables



```java
//refreshes every table in the Netbeans GUI by taking in data from MySQL
public void refreshAllGUITables(){
    refreshGUITable(fridge1Table, "cs_ia.fridge_1", storageOrder);
    refreshGUITable(fridge2Table, "cs_ia.fridge_2", storageOrder);
    refreshGUITable(freezer1Table, "cs_ia.freezer_1", storageOrder);
    refreshGUITable(freezer2Table, "cs_ia.freezer_2", storageOrder);
    refreshGUITable(ingredientsTable, "cs_ia.ingredient_bases", ingredientsOrder);
    refreshGUITable(recipesTable, "cs_ia.recipes", recipesOrder);
    refreshGUITable(suggestionsTable, "cs_ia.meal_suggestions", suggestionsOrder);
    refreshGUITable(expiredTable, "cs_ia.expired_foods", expiredOrder);
    refreshGUITable(expiringTable, "cs_ia.expiring_foods", expiringOrder);
}
```

refreshAllGUITables() calls on **refreshGUITable()** 9 times, once for every JTable in the GUI. This was done to avoid clutter, as I could call a single method instead of 9 every time I wanted to refresh the GUI.

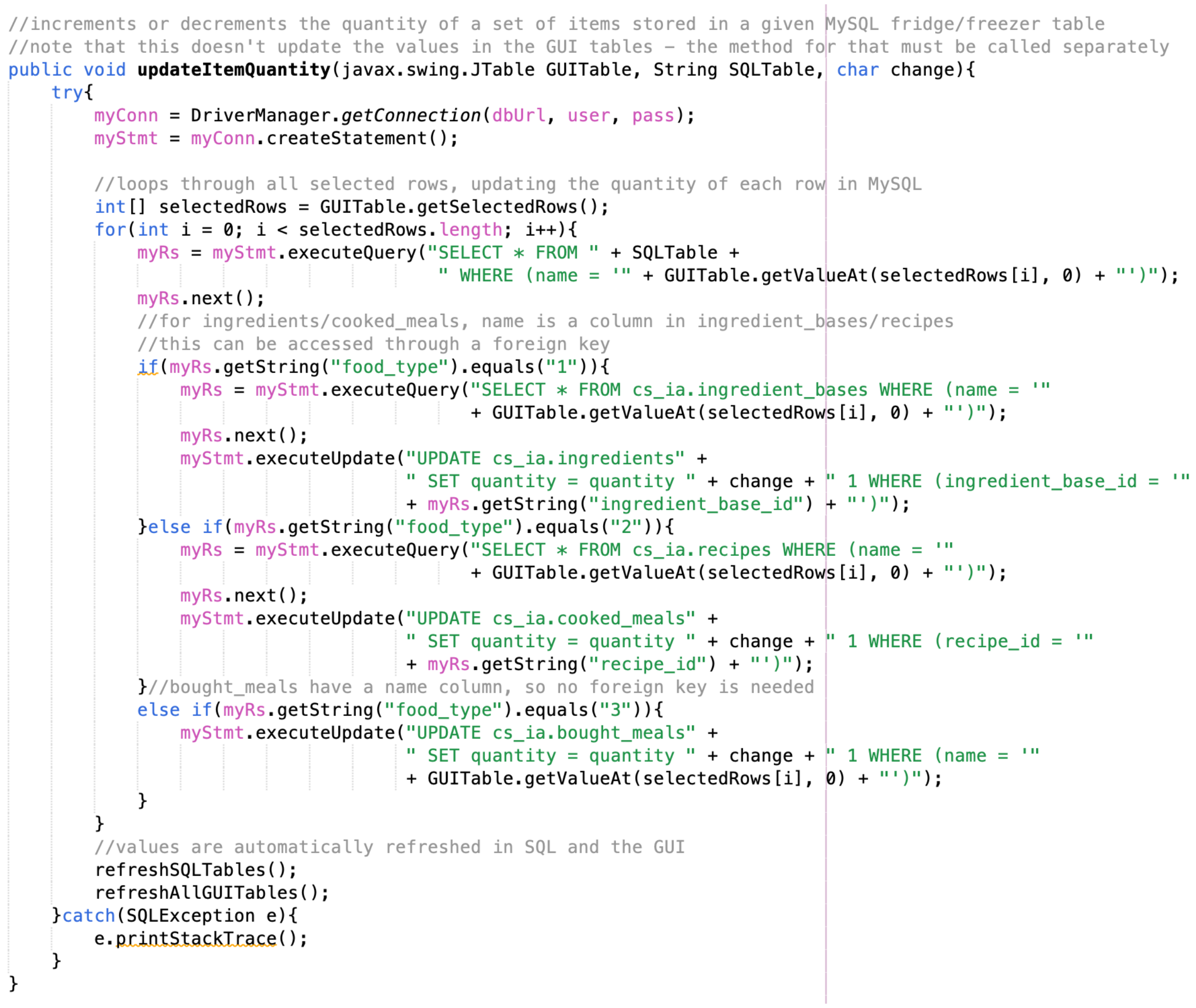
Psuedocode for **refreshGUITable()**:

```
refreshGUITable(parameters: GUI_TABLE, SQL_TABLE, SORTING_ORDER)
        connect to MySQL database
        select all rows from SQL_TABLE
        clear rows from GUI_TABLE
        if SQL_TABLE is ingredient_bases
                insert rows into GUI_TABLE with ‘name’ and ‘type’ data from SQL_TABLE sorted…
                … by SORTING_ORDER
        else if SQL_TABLE is recipes or meal_suggestions
                convert SQL_TABLE’s ‘ingredients_list’ data to string INGREDIENTS
                insert rows into GUI_TABLE with ‘name’ (from SQL_TABLE) and INGREDIENTS…
                … sorted by SORTING_ORDER
        else if SQL_TABLE has 3 columns
                insert rows into GUI_TABLE with ‘name’, ‘quantity’, and ‘expiration_date’…
                … data from SQL_TABLE sorted by SORTING_ORDER
        end if
end method
```

The most complicated aspect of **refreshGUITable()** was converting ingredients_list, which is a String of keys separated by commas (e.g. “1,5,2”), to a more readable String (e.g. “Ham, Cheese, Bread”). The code for this is shown below:

```
String ingredientIDs = myRs.getString("ingredients_list");
LinkedList<String> IDsList = new LinkedList<String>();
//example of what ingredientIDs may look like: "4,1,12,8,15"

String ingredients = "";
String temp = "";

//populates IDsList with the numbers separated by commas in ingredientIDs
//with the same example, the first element of IDsList would be "4" and the third would be "12"
for(int i = 0; i < ingredientIDs.length(); i++){
    if(ingredientIDs.charAt(i) != ','){
        temp += ingredientIDs.charAt(i);
    }else{
        IDsList.add(temp);
        temp = "";
    }
}
IDsList.add(temp);

//finds the names of the ingredient_base for each element of IDsList
for(int i = 0; i < IDsList.size(); i++){
    myRs2 = myStmt2.executeQuery("SELECT * FROM ingredient_bases WHERE ingredient_base_id = '" + IDsList.get(i) + "';");
    myRs2.next();
    ingredients += myRs2.getString("name");
    ingredients += ", ";
}
ingredients = ingredients.substring(0, ingredients.length() -2);
//ingredients may look something like "Ham, Cheese, Bread"
//this is much more useful for the user than displaying the IDs of these ingredients in the form "3,5,1"
```

4. Changing the quantity of stored item



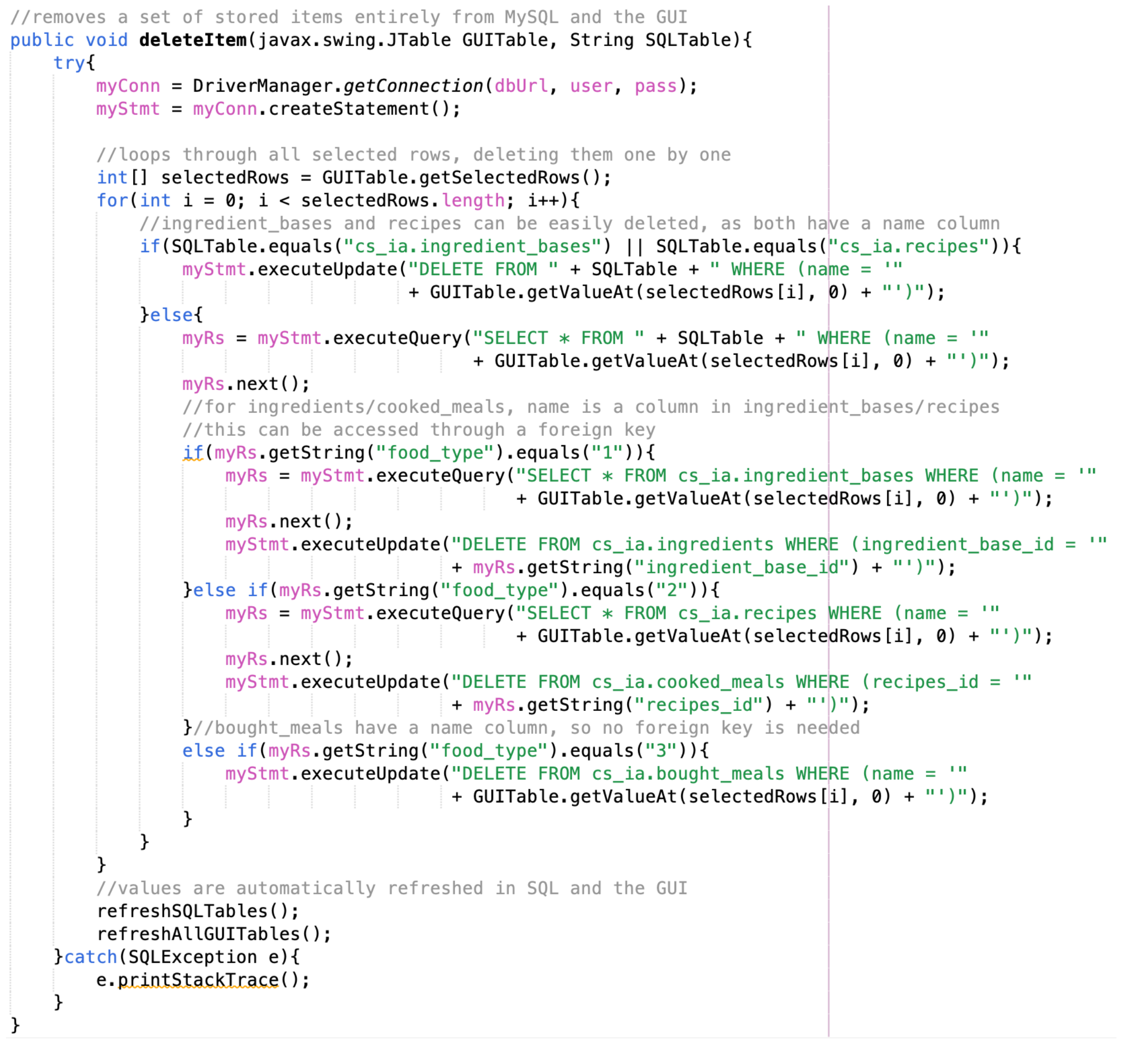
```java
//increments or decrements the quantity of a set of items stored in a given MySQL fridge/freezer table
//note that this doesn't update the values in the GUI tables - the method for that must be called separately
public void updateItemQuantity(javax.swing.JTable GUITable, String SQLTable, char change){
    try{
        myConn = DriverManager.getConnection(dbUrl, user, pass);
        myStmt = myConn.createStatement();

        //loops through all selected rows, updating the quantity of each row in MySQL
        int[] selectedRows = GUITable.getSelectedRows();
        for(int i = 0; i < selectedRows.length; i++){
            myRs = myStmt.executeQuery("SELECT * FROM " + SQLTable +
                                       " WHERE (name = '" + GUITable.getValueAt(selectedRows[i], 0) + "')");
            myRs.next();
            //for ingredients/cooked_meals, name is a column in ingredient_bases/recipes
            //this can be accessed through a foreign key
            if(myRs.getString("food_type").equals("1")){
                myRs = myStmt.executeQuery("SELECT * FROM cs_ia.ingredient_bases WHERE (name = '"
                                           + GUITable.getValueAt(selectedRows[i], 0) + "')");
                myRs.next();
                myStmt.executeUpdate("UPDATE cs_ia.ingredients" +
                                     " SET quantity = quantity " + change + " 1 WHERE (ingredient_base_id = '"
                                     + myRs.getString("ingredient_base_id") + "')");
            }else if(myRs.getString("food_type").equals("2")){
                myRs = myStmt.executeQuery("SELECT * FROM cs_ia.recipes WHERE (name = '"
                                           + GUITable.getValueAt(selectedRows[i], 0) + "')");
                myRs.next();
                myStmt.executeUpdate("UPDATE cs_ia.cooked_meals" +
                                     " SET quantity = quantity " + change + " 1 WHERE (recipe_id = '"
                                     + myRs.getString("recipe_id") + "')");
            }//bought_meals have a name column, so no foreign key is needed
            else if(myRs.getString("food_type").equals("3")){
                myStmt.executeUpdate("UPDATE cs_ia.bought_meals" +
                                     " SET quantity = quantity " + change + " 1 WHERE (name = '"
                                     + GUITable.getValueAt(selectedRows[i], 0) + "')");
            }
        }
        //values are automatically refreshed in SQL and the GUI
        refreshSQLTables();
        refreshAllGUITables();
    }catch(SQLException e){
        e.printStackTrace();
    }
}
```

For each row selected in GUITable by the user, **updateItemQuantity()** finds the corresponding ingredient/cooked_meal/bought_meal in MySQL and increments or decrements its quantity.

Initially, I made the mistake of changing the quantity in the SQL fridge/freezer where this food was stored. The problem was that, whenever **refreshSQLTables()** was called, this fridge/freezer would be updated with values from ingredients, cooked_meals, and bought_meals, and the quantity would change back to what it was before. Thus, I had to first change quantity in ingredients/cooked_meals/bought_meals before updating the storages. This is what the "food_type" selection clauses are for.

Also, since ingredients and cooked_meals don't have a "name" column, I first had to find the ingredient_base/recipe with the correct name, before finding the ingredient/cooked_meal with the same ingredient_base_id/recipe_id FK.

5. Deleting an item



```java
//removes a set of stored items entirely from MySQL and the GUI
public void deleteItem(javax.swing.JTable GUITable, String SQLTable){
    try{
        myConn = DriverManager.getConnection(dbUrl, user, pass);
        myStmt = myConn.createStatement();

        //loops through all selected rows, deleting them one by one
        int[] selectedRows = GUITable.getSelectedRows();
        for(int i = 0; i < selectedRows.length; i++){
            //ingredient_bases and recipes can be easily deleted, as both have a name column
            if(SQLTable.equals("cs_ia.ingredient_bases") || SQLTable.equals("cs_ia.recipes")){
                myStmt.executeUpdate("DELETE FROM " + SQLTable + " WHERE (name = '"
                                    + GUITable.getValueAt(selectedRows[i], 0) + "')");
            }else{
                myRs = myStmt.executeQuery("SELECT * FROM " + SQLTable + " WHERE (name = '"
                                        + GUITable.getValueAt(selectedRows[i], 0) + "')");
                myRs.next();
                //for ingredients/cooked_meals, name is a column in ingredient_bases/recipes
                //this can be accessed through a foreign key
                if(myRs.getString("food_type").equals("1")){
                    myRs = myStmt.executeQuery("SELECT * FROM cs_ia.ingredient_bases WHERE (name = '"
                                            + GUITable.getValueAt(selectedRows[i], 0) + "')");
                    myRs.next();
                    myStmt.executeUpdate("DELETE FROM cs_ia.ingredients WHERE (ingredient_base_id = '"
                                        + myRs.getString("ingredient_base_id") + "')");
                }else if(myRs.getString("food_type").equals("2")){
                    myRs = myStmt.executeQuery("SELECT * FROM cs_ia.recipes WHERE (name = '"
                                            + GUITable.getValueAt(selectedRows[i], 0) + "')");
                    myRs.next();
                    myStmt.executeUpdate("DELETE FROM cs_ia.cooked_meals WHERE (recipes_id = '"
                                        + myRs.getString("recipes_id") + "')");
                }//bought_meals have a name column, so no foreign key is needed
                else if(myRs.getString("food_type").equals("3")){
                    myStmt.executeUpdate("DELETE FROM cs_ia.bought_meals WHERE (name = '"
                                        + GUITable.getValueAt(selectedRows[i], 0) + "')");
                }
            }
        }
        //values are automatically refreshed in SQL and the GUI
        refreshSQLTables();
        refreshAllGUITables();
    }catch(SQLException e){
        e.printStackTrace();
    }
}
```

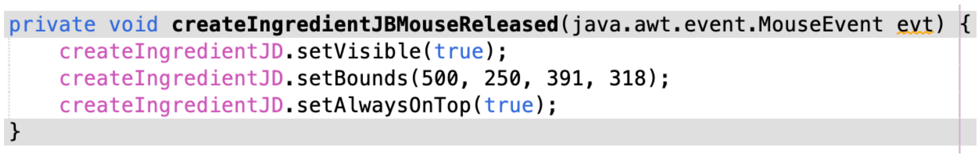
deleteItem() works very similarly to **updateItemQuantity()**, except that ingredient_bases and recipes can also be deleted.

Word Count: 292 (excluding bulleted lists)

Other Parts of the Code

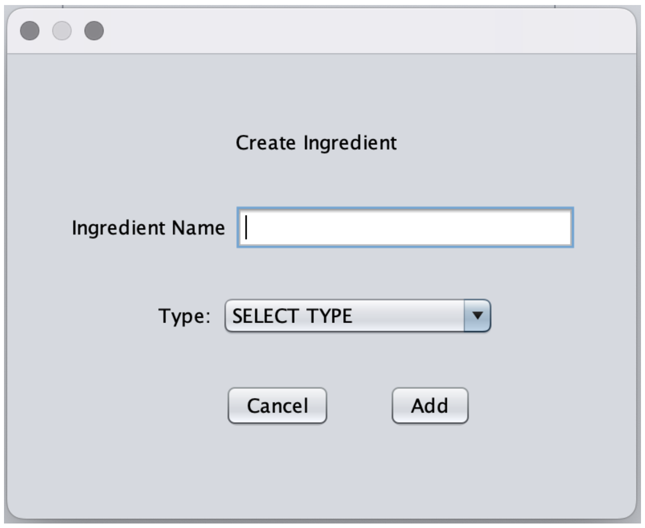
Most of the code happens when MouseListeners are activated. A lot of this is repetitive, but I'll explain some examples.

1. Create button

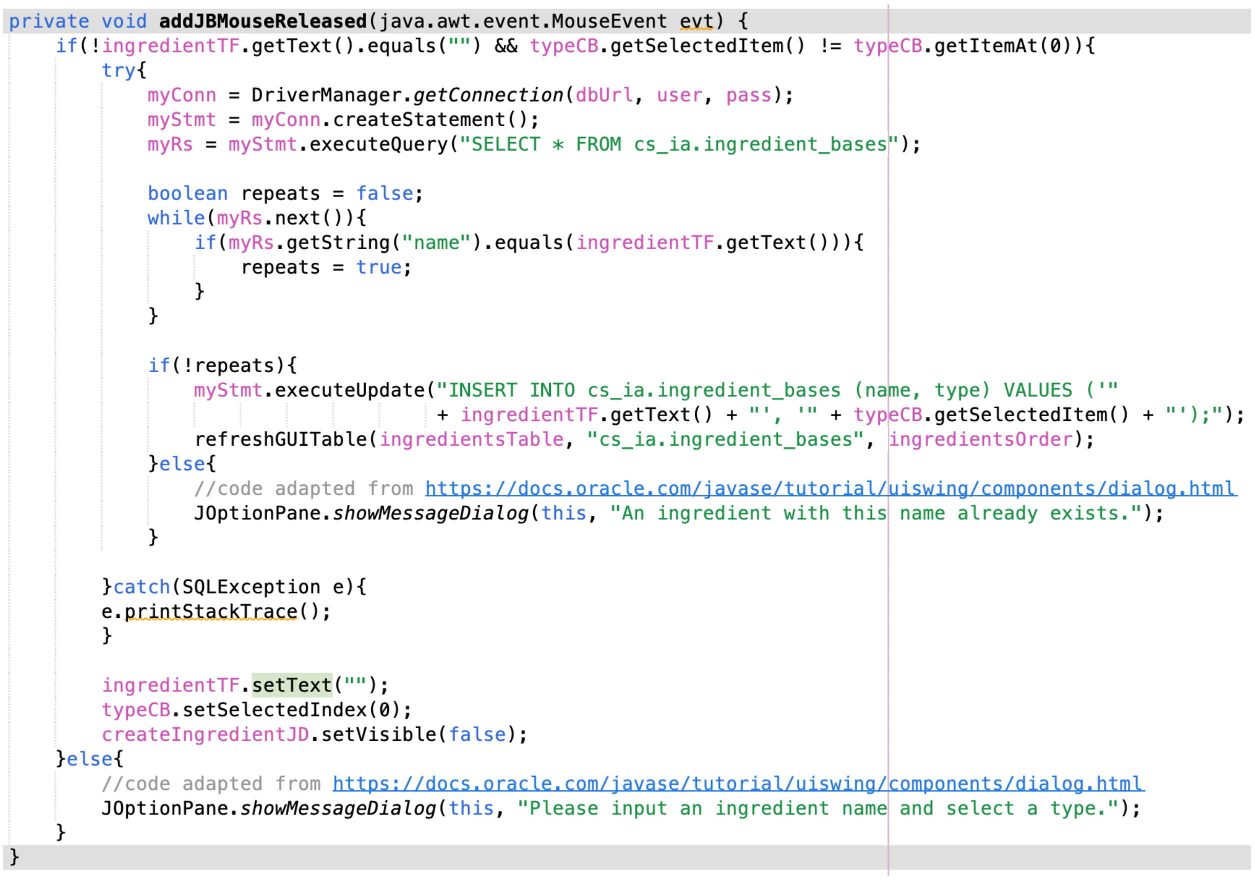


```
private void createIngredientJBMouseReleased(java.awt.event.MouseEvent evt) {
    createIngredientJD.setVisible(true);
    createIngredientJD.setBounds(500, 250, 391, 318);
    createIngredientJD.setAlwaysOnTop(true);
}
```

Clicking “Create New Ingredient” in the “Recipes/Ingredients” tab executes this code, which opens a JDialog.

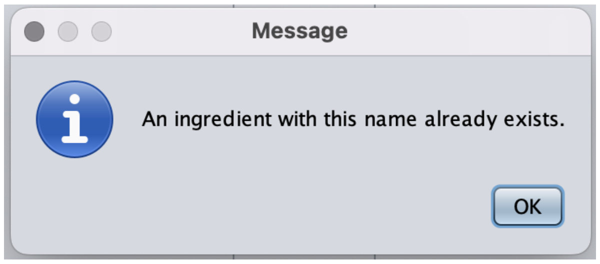


2. Add button



```java
private void addJBMouseReleased(java.awt.event.MouseEvent evt) {
    if(!ingredientTF.getText().equals("") && typeCB.getSelectedItem() != typeCB.getItemAt(0)){
        try{
            myConn = DriverManager.getConnection(dbUrl, user, pass);
            myStmt = myConn.createStatement();
            myRs = myStmt.executeQuery("SELECT * FROM cs_ia.ingredient_bases");

            boolean repeats = false;
            while(myRs.next()){
                if(myRs.getString("name").equals(ingredientTF.getText())){
                    repeats = true;
                }
            }

            if(!repeats){
                myStmt.executeUpdate("INSERT INTO cs_ia.ingredient_bases (name, type) VALUES ('"
                                      + ingredientTF.getText() + "', '" + typeCB.getSelectedItem() + "');");
                refreshGUITable(ingredientsTable, "cs_ia.ingredient_bases", ingredientsOrder);
            }else{
                //code adapted from https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html
                JOptionPane.showMessageDialog(this, "An ingredient with this name already exists.");
            }

        }catch(SQLException e){
        e.printStackTrace();
        }

        ingredientTF.setText("");
        typeCB.setSelectedIndex(0);
        createIngredientJD.setVisible(false);
    }else{
        //code adapted from https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html
        JOptionPane.showMessageDialog(this, "Please input an ingredient name and select a type.");
    }
}
```

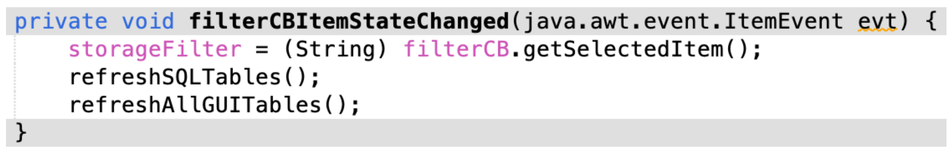
Clicking "Add" in the aforementioned JDialog takes in user input and inserts a new row into the MySQL ingredient_bases table. This code checks to make sure all input boxes have been filled out, and the ingredient_base name doesn't already exists, displaying an error message if either of these isn't true[4].



An ingredient with this name already exists.

3. Storage filter

[4] The code for opening a JOptionPane message dialog was adapted from an online source. Citation: *How to Make Dialogs*. (n.d.). Oracle. https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html



```java
private void filterCBItemStateChanged(java.awt.event.ItemEvent evt) {
    storageFilter = (String) filterCB.getSelectedItem();
    refreshSQLTables();
    refreshAllGUITables();
}
```

The filter combo box lets the user pick which filter they want to apply into items in the storage tables. Changing the filter will change the state of **filterCB**, and this method will run, changing the global filtering variable for storages and refreshing all tables so the filtered information is seen.

<u>Word Count: 139</u> (excluding bulleted lists)