Criterion C: Development

Table of Contents

Key Features Used

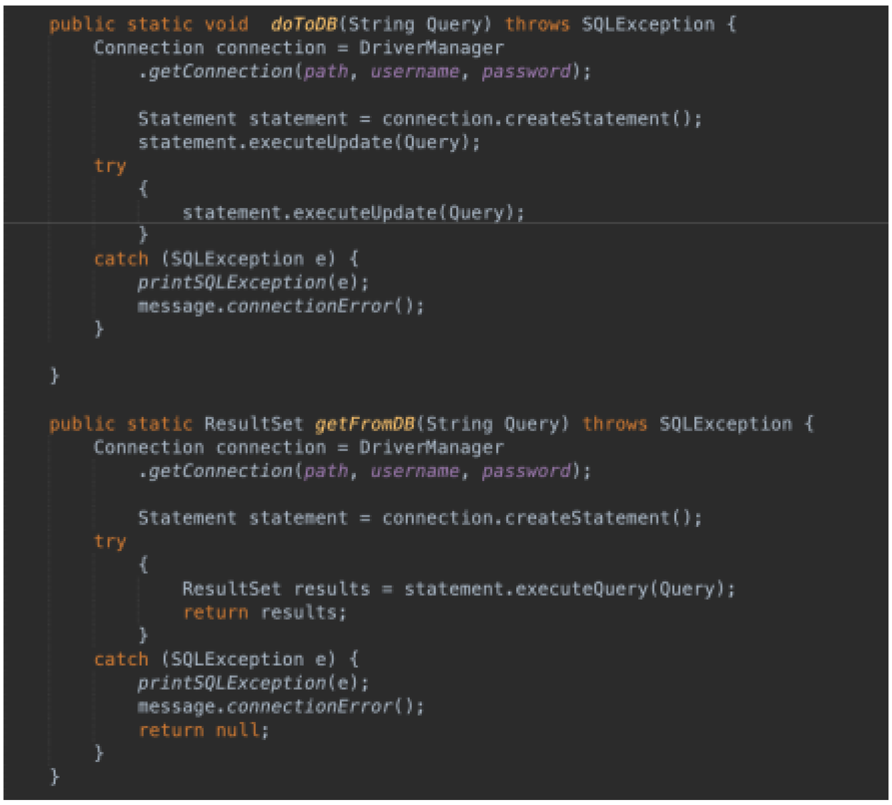
	Feature	**Purpose/Value**
1	Complex SQL queries with Java	This was used to insert, delete and update records in the Grocery System database as well as get results from the database with specific conditions. Many of the queries used, combine multiple tables to display information in one result set. Furthermore, many of the queries have user-defined variables. This helps meet CFS 1, 2, 3, 4, 5, 6, 7 and 8.
2	Event handling	Many of the programs are linked to user actions, such as clicking a certain button or selecting a row on the table. This is done with event handling.
3	Inputs from jTables	Users were able to input information through the jTable in the form of checkboxes and cell inputs. Users could toggle checkboxes box on or off, and this was used accordingly to either delete or select multiple items at once. This feature helps meet CFS 2b, and 8b.
4	Writing on the text file	The shopping list can be written on a text file in the user's desktop, which meets CFS 12.
5	Error handling	Error handling was used to make the system more

		user-friendly. If the system was not working as intended, the error handling would flag the issue to the user in an understandable way.
6	2D arraysList with nested loops	The 2D array list was used to store information that was retrieved from a query search, and the nested loops would search through the 2D array to merge the information of duplicate items. This helps meet CFS 1 , 2, and 6.
7	Validation of all inputs	All inputs have had several checks run (depending on what type of data is required) on them to ensure that the data being entered is valid.
8	Additional libraries	Additional java libraries were used to add charts to the application. This helps meet CFS 10 and 11.

1. SQL techniques

1. Connection to and operations with the database

Two methods were written that would connect to the database using the java.sql package.

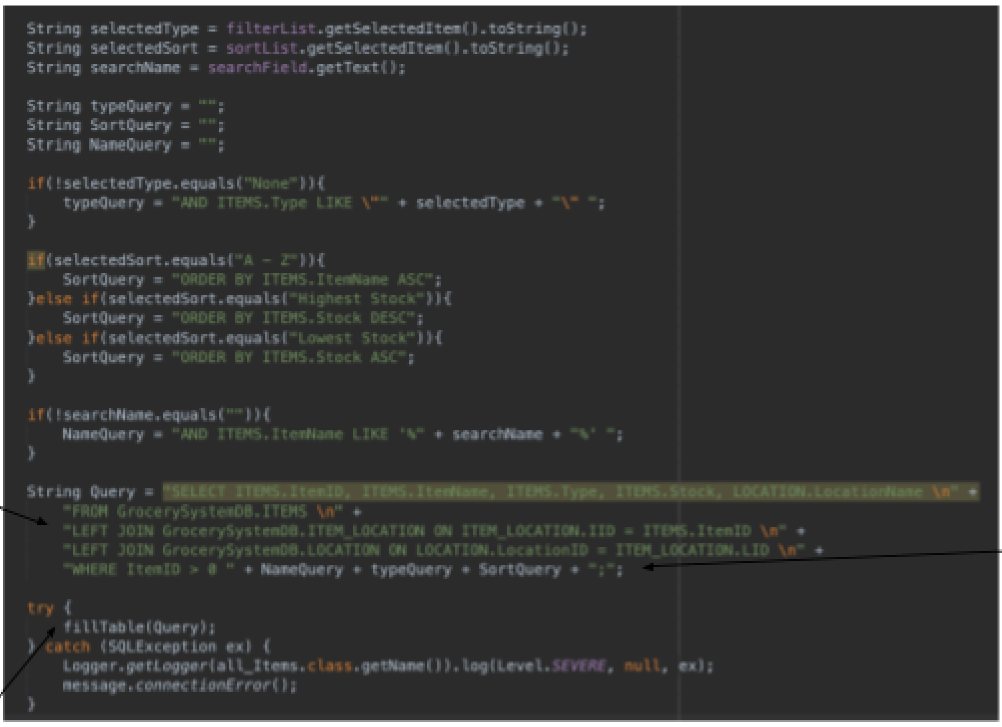


```
public static void  doToDB(String Query) throws SQLException {
    Connection connection = DriverManager
        .getConnection(path, username, password);

        Statement statement = connection.createStatement();
        statement.executeUpdate(Query);
    try
        {
            statement.executeUpdate(Query);
        }
    catch (SQLException e) {
        printSQLException(e);
        message.connectionError();
    }

}

public static ResultSet getFromDB(String Query) throws SQLException {
    Connection connection = DriverManager
        .getConnection(path, username, password);

        Statement statement = connection.createStatement();
    try
        {
            ResultSet results = statement.executeQuery(Query);
            return results;
        }
    catch (SQLException e) {
        printSQLException(e);
        message.connectionError();
        return null;
    }
}
```

doToDB executes queries where there is no result retrieved, and getFromDB executes queries where data needs to be retrieved in the form of a Result Set.

2. Complex queries implemented with java

Information presented requires joining multiple tables together. The java used, adds conditions to the query when required.



```
String selectedType = filterList.getSelectedItem().toString();
String selectedSort = sortList.getSelectedItem().toString();
String searchName = searchField.getText();

String typeQuery = "";
String SortQuery = "";
String NameQuery = "";

if(!selectedType.equals("None")){
    typeQuery = "AND ITEMS.Type LIKE \"" + selectedType + "\" ";
}

if(selectedSort.equals("A - Z")){
    SortQuery = "ORDER BY ITEMS.ItemName ASC";
}else if(selectedSort.equals("Highest Stock")){
    SortQuery = "ORDER BY ITEMS.Stock DESC";
}else if(selectedSort.equals("Lowest Stock")){
    SortQuery = "ORDER BY ITEMS.Stock ASC";
}

if(!searchName.equals("")){
    NameQuery = "AND ITEMS.ItemName LIKE '%" + searchName + "%' ";
}

String Query = "SELECT ITEMS.ItemID, ITEMS.ItemName, ITEMS.Type, ITEMS.Stock, LOCATION.LocationName \n" +
    "FROM GrocerySystemDB.ITEMS \n" +
    "LEFT JOIN GrocerySystemDB.ITEM_LOCATION ON ITEM_LOCATION.IID = ITEMS.ItemID \n" +
    "LEFT JOIN GrocerySystemDB.LOCATION ON LOCATION.LocationID = ITEM_LOCATION.LID \n" +
    "WHERE ItemID > 0 " + NameQuery + typeQuery + SortQuery + ";";

try {
    fillTable(Query);
} catch (SQLException ex) {
    Logger.getLogger(all_Items.class.getName()).log(Level.SEVERE, null, ex);
    message.connectionError();
}
```

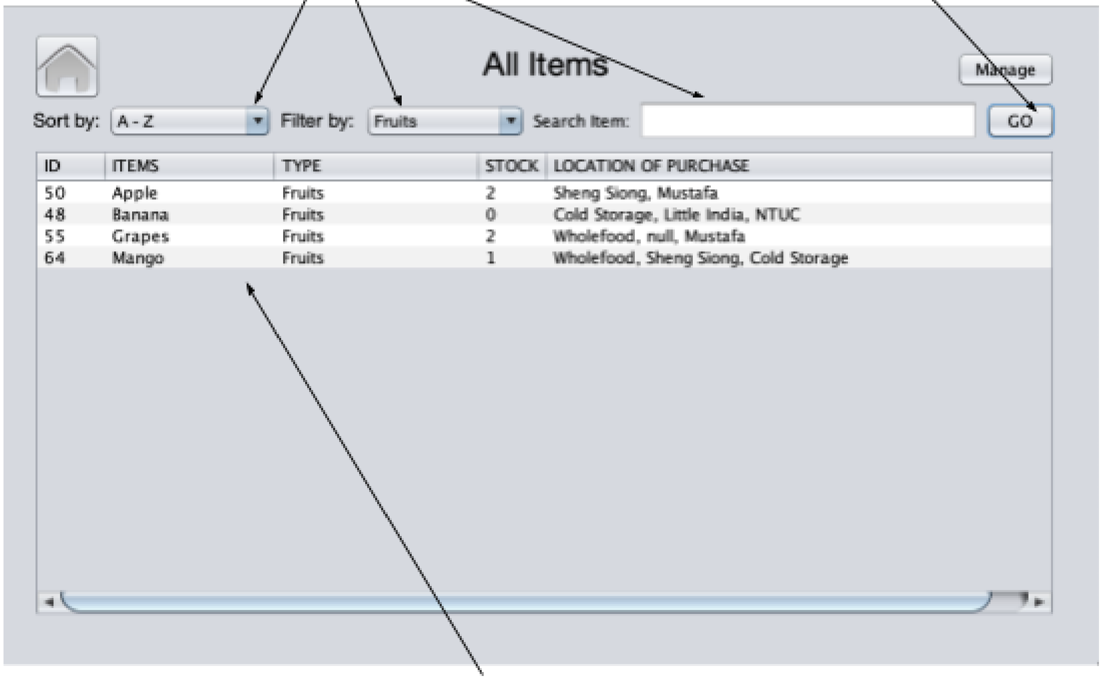
Query with joining multiple tables together

combining java strings to set conditions for the query

The fill table query uses

when filled, they go through the if statement, and the conditions get added to the query

On pressing the button, the code with the sql command is run



ID	ITEMS	TYPE	STOCK	LOCATION OF PURCHASE
50	Apple	Fruits	2	Sheng Siong, Mustafa
48	Banana	Fruits	0	Cold Storage, Little India, NTUC
55	Grapes	Fruits	2	Wholefood, null, Mustafa
64	Mango	Fruits	1	Wholefood, Sheng Siong, Cold Storage

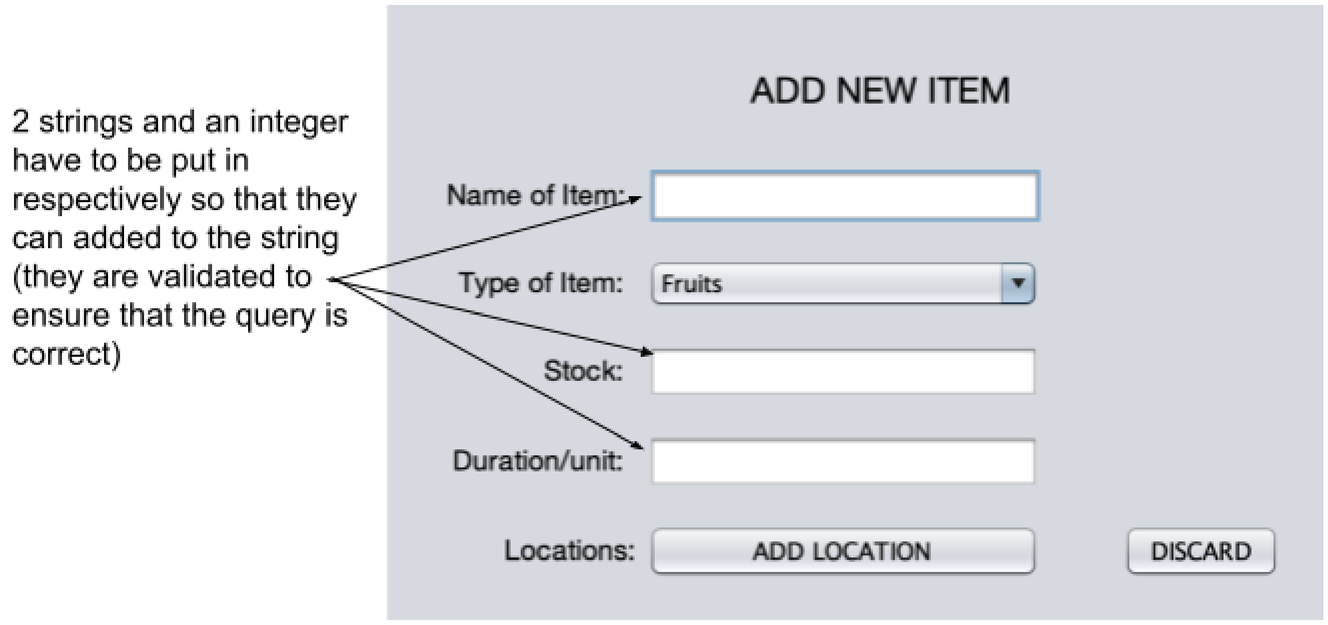
The table is filled with items returned from the query

This meets CFS 3, 4, and 5.

The inputs from the two drop-down boxes and the textbox will get the conditions for the sorting, filtering and searching, and add them to the query.

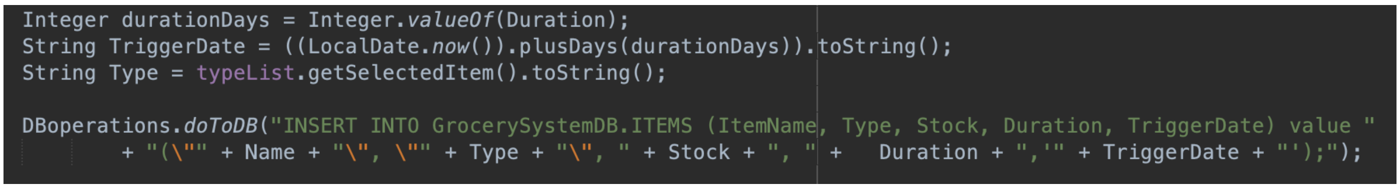
Ingenuity:
This allows the user to view specific items that they want to see. They can use the sort, filter or search feature to find the exact item/s they want. It also allows the user to find all the items that need to be bought when going to several locations at once, instead of having to create multiple lists.

3. Inserting items into the database



2 strings and an integer have to be put in respectively so that they can added to the string (they are validated to ensure that the query is correct)

The inputs from the textboxes are validated and then added to the query, meeting CFS 2a.

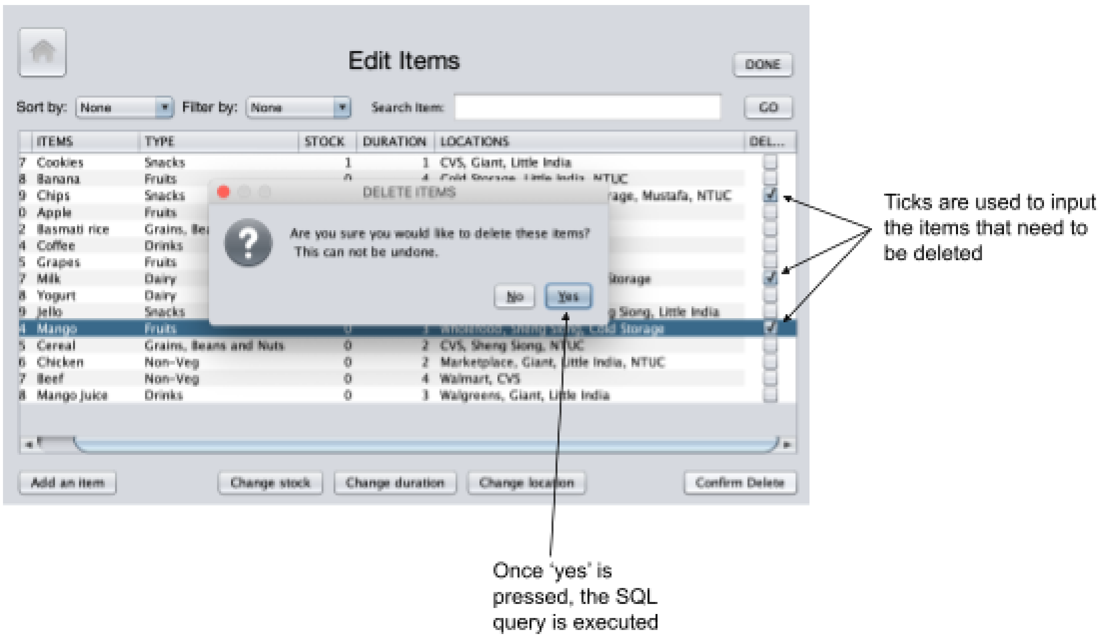


```
Integer durationDays = Integer.valueOf(Duration);
String TriggerDate = ((LocalDate.now()).plusDays(durationDays)).toString();
String Type = typeList.getSelectedItem().toString();

DBoperations.doToDB("INSERT INTO GrocerySystemDB.ITEMS (ItemName, Type, Stock, Duration, TriggerDate) value "
        + "(\"" + Name + "\", \"" + Type + "\", " + Stock + ", " +   Duration + ",'" + TriggerDate + "');");
```

Similar code is used to add locations.

4. Deleting items from the database

Users can choose items to delete based on the items they select. Deletion is done using the itemID of the item, meeting CFS 2b and 8b.


Ticks are used to input the items that need to be deleted
Once 'yes' is pressed, the SQL query is executed

Edit Items

DONE

Sort by: None Filter by: None Search item: GO

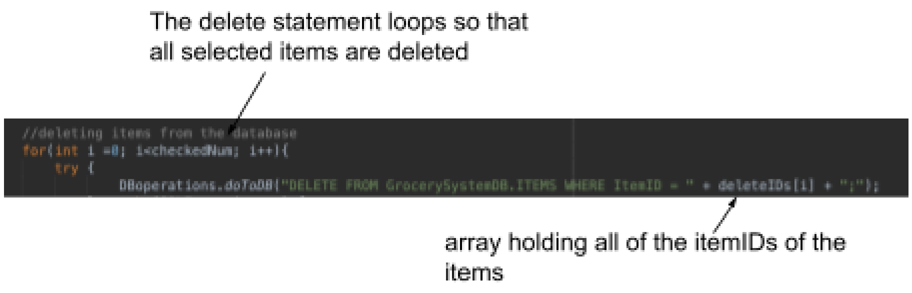
	ITEMS	TYPE	STOCK	DURATION	LOCATIONS	DEL...
7	Cookies	Snacks	1	1	CVS, Giant, Little India	☐
8	Banana	Fruits	0	4	Cold Storage, Little India, NTUC	☐
9	Chips	Snacks			rage, Mustafa, NTUC	✓
0	Apple	Fruits				☐
2	Basmati rice	Grains, Be				☐
4	Coffee	Drinks				☐
5	Grapes	Fruits				☐
7	Milk	Dairy			Storage	✓
8	Yogurt	Dairy				☐
9	Jello	Snacks			g Siong, Little India	☐
4	Mango	Fruits	0	3	Wholefoods, Sheng Siong, Cold Storage	✓
5	Cereal	Grains, Beans and Nuts	0	2	CVS, Sheng Siong, NTUC	☐
6	Chicken	Non-Veg	0	2	Marketplace, Giant, Little India, NTUC	☐
7	Beef	Non-Veg	0	4	Walmart, CVS	☐
8	Mango Juice	Drinks	0	3	Walgreens, Giant, Little India	☐

DELETE ITEMS

Are you sure you would like to delete these items?
This can not be undone.

No Yes

Add an item Change stock Change duration Change location Confirm Delete


The delete statement loops so that all selected items are deleted
array holding all of the itemIDs of the items

```
//deleting items from the database
for(int i =0; i<checkedNum; i++){
    try {
        DBoperations.doToDB("DELETE FROM GrocerySystemDB.ITEMS WHERE ItemID = " + deleteIDs[i] + ";");
```

5. Updating the database

Updating the database is used when the system automatically updates the stock of an item on its trigger date.

```
DBoperations.doToDB("UPDATE GrocerySystemDB.ITEMS " +
        "SET Stock = "+ stock +", TriggerDate = '" + tempDate + "' WHERE ItemID = " + ID + ";");
```

Items are also updated when the user changes information on the item (CFS 2c, 8a). The update query allows the implementation of java variables after validation to change information in the database.

2. Event handling

Events are programmed to respond after a certain user clicks. This was implemented using ActionEvent which extends AWTEvent.

Example of button press:

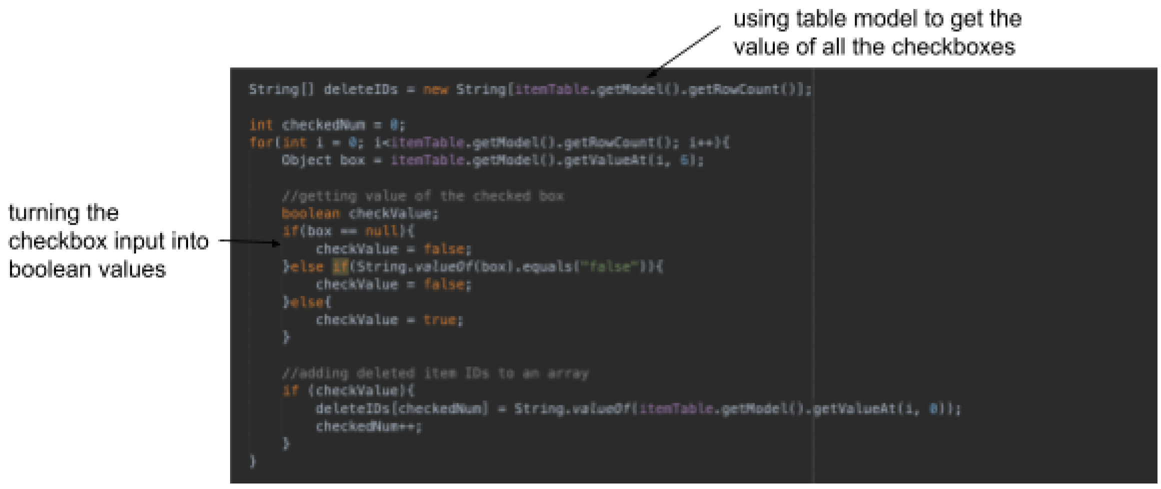
```
goButton.setText("GO");
goButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        goButtonActionPerformed(evt);
    }
});
```

```
private void goButtonActionPerformed(java.awt.event.ActionEvent evt) {
```

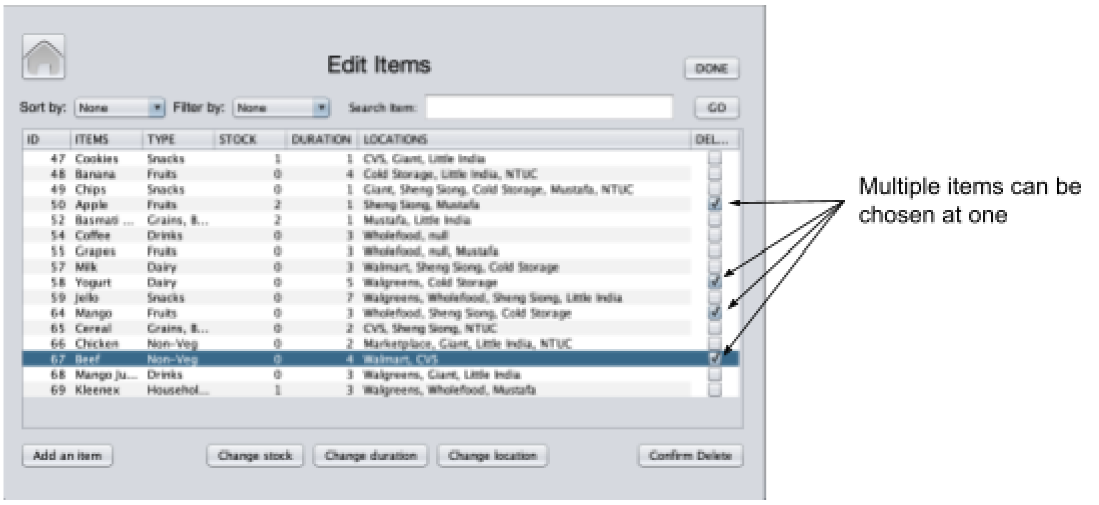
3. Inputs from jTables

The jTable also takes in user inputs by retrieving the table model.

Using checkboxes to delete multiple items at once:


using table model to get the value of all the checkboxes
turning the checkbox input into boolean values

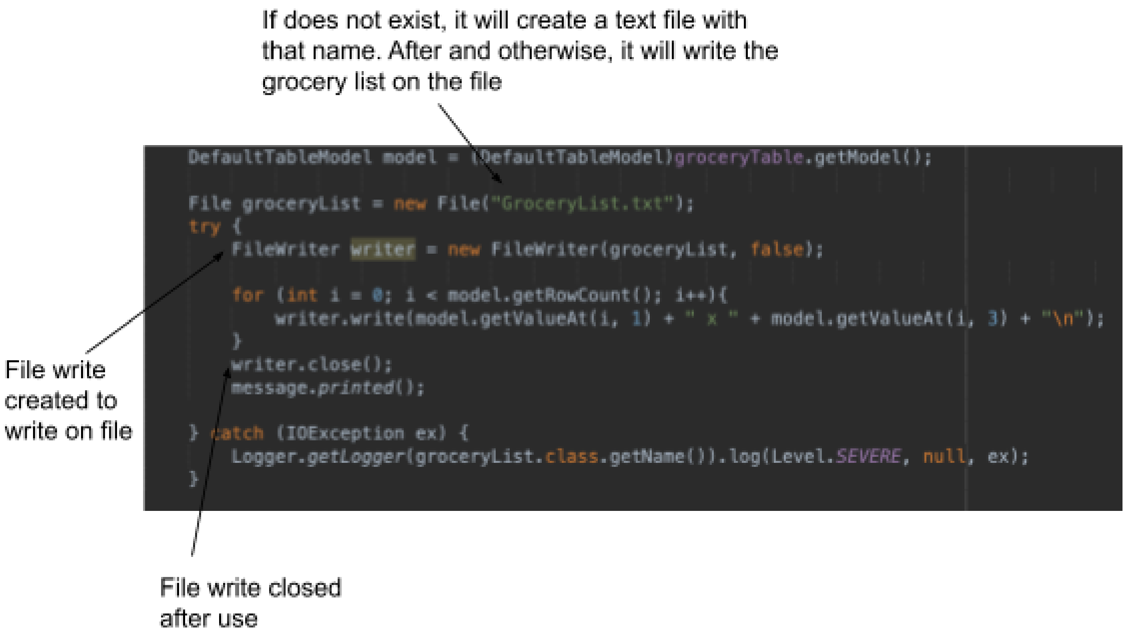
```
String[] deleteIDs = new String[itemTable.getModel().getRowCount()];

int checkedNum = 0;
for(int i = 0; i<itemTable.getModel().getRowCount(); i++){
    Object box = itemTable.getModel().getValueAt(i, 6);

    //getting value of the checked box
    boolean checkValue;
    if(box == null){
        checkValue = false;
    }else if(String.valueOf(box).equals("false")){
        checkValue = false;
    }else{
        checkValue = true;
    }

    //adding deleted item IDs to an array
    if (checkValue){
        deleteIDs[checkedNum] = String.valueOf(itemTable.getModel().getValueAt(i, 0));
        checkedNum++;
    }
}
```



ID	ITEMS	TYPE	STOCK	DURATION	LOCATIONS	DEL...
47	Cookies	Snacks	1	1	CVS, Giant, Little India	☐
48	Banana	Fruits	0	4	Cold Storage, Little India, NTUC	☐
49	Chips	Snacks	0	1	Giant, Sheng Siong, Cold Storage, Mustafa, NTUC	☐
50	Apple	Fruits	2	1	Sheng Siong, Mustafa	✓
52	Basmati ...	Grains, B...	2	1	Mustafa, Little India	☐
54	Coffee	Drinks	0	3	Wholefood, null	☐
55	Grapes	Fruits	0	3	Wholefood, null, Mustafa	☐
57	Milk	Dairy	0	3	Walmart, Sheng Siong, Cold Storage	☐
58	Yogurt	Dairy	0	5	Walgreens, Cold Storage	✓
59	Jello	Snacks	0	7	Walgreens, Wholefood, Sheng Siong, Little India	☐
64	Mango	Fruits	0	3	Wholefood, Sheng Siong, Cold Storage	✓
65	Cereal	Grains, B...	0	2	CVS, Sheng Siong, NTUC	☐
66	Chicken	Non-Veg	0	2	Marketplace, Giant, Little India, NTUC	☐
67	Beef	Non-Veg	0	4	Walmart, CVS	✓
68	Mango ju...	Drinks	0	3	Walgreens, Giant, Little India	☐
69	Kleenex	Househol...	1	3	Walgreens, Wholefood, Mustafa	☐

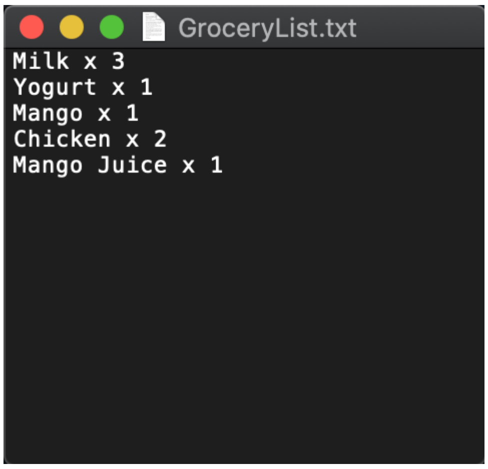
<u>Ingenuity:</u>
Using the table to take user inputs allows the client to do an operation on multiple items at one time instead of having to do them one by one.

4. Writing on the text file


If does not exist, it will create a text file with that name. After and otherwise, it will write the grocery list on the file
File write created to write on file
File write closed after use

```java
DefaultTableModel model = (DefaultTableModel)groceryTable.getModel();

File groceryList = new File("GroceryList.txt");
try {
    FileWriter writer = new FileWriter(groceryList, false);

    for (int i = 0; i < model.getRowCount(); i++){
        writer.write(model.getValueAt(i, 1) + " x " + model.getValueAt(i, 3) + "\n");
    }
    writer.close();
    message.printed();

} catch (IOException ex) {
    Logger.getLogger(groceryList.class.getName()).log(Level.SEVERE, null, ex);
}
```

The system can create a text file with the user's grocery list that they can print. The text file would be created if it did not already exist.



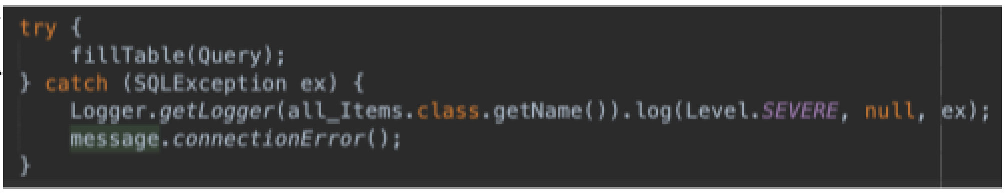
```
Milk x 3
Yogurt x 1
Mango x 1
Chicken x 2
Mango Juice x 1
```

This meets CFS 12

Ingenuity:
Printing the shopping list on a separate file allows the user to copy and paste it somewhere else (in an email or a text) or even print it out on paper. Without this, the client would have to manually write or retype the list somewhere else.

5. Error handling

The program throws an SQLException when there is an issue with connecting to the database. try...catch statements were used to tell the client in a user-friendly way what the error is.

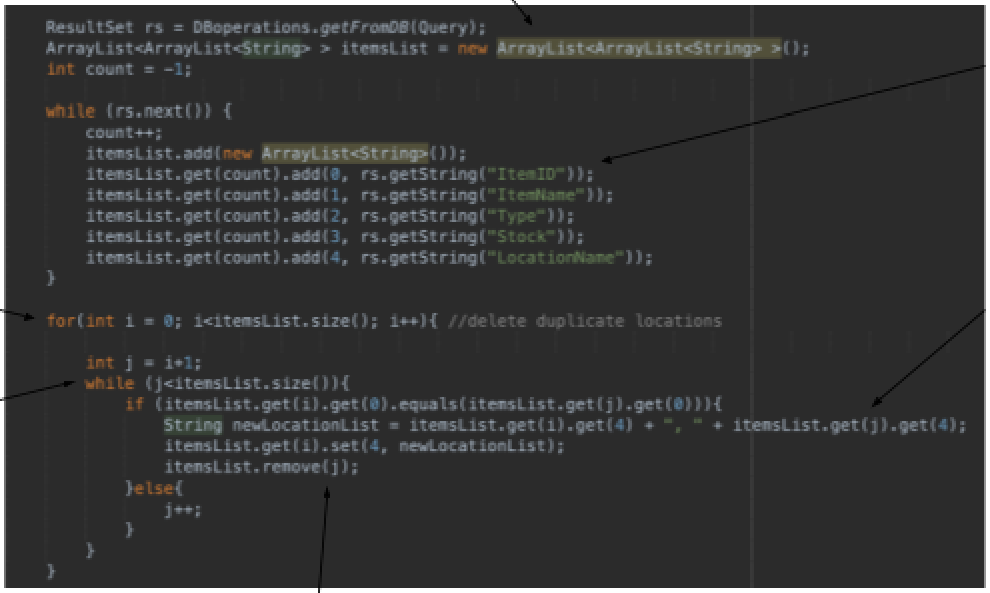

action that could potentially throw a SQLException is surrounded by try…catch statement

```
try {
    fillTable(Query);
} catch (SQLException ex) {
    Logger.getLogger(all_Items.class.getName()).log(Level.SEVERE, null, ex);
    message.connectionError();
}
```

Ingenuity:
Because the exceptions are caught, instead of showing the user a complicated message that would be difficult to understand, the error is changed to a simplified message.

6. 2D arraysList with nested loops

A 2D arraysList with nested FOR loops was used when viewing items with multiple corresponding locations.

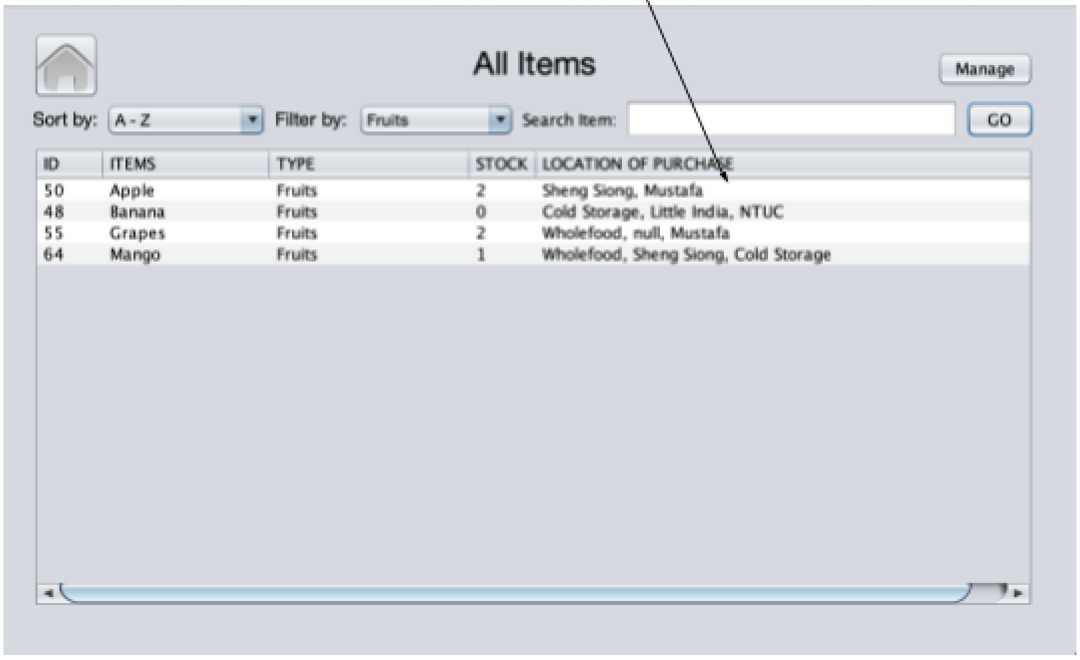

Creation of 2D string arrayList
Adding to the 2D arrayList
First for loop
retrieving from the arrayList
While loop nested inside for loop
removing items from the arrayList

```
ResultSet rs = DBoperations.getFromDB(Query);
ArrayList<ArrayList<String> > itemsList = new ArrayList<ArrayList<String> >();
int count = -1;

while (rs.next()) {
    count++;
    itemsList.add(new ArrayList<String>());
    itemsList.get(count).add(0, rs.getString("ItemID"));
    itemsList.get(count).add(1, rs.getString("ItemName"));
    itemsList.get(count).add(2, rs.getString("Type"));
    itemsList.get(count).add(3, rs.getString("Stock"));
    itemsList.get(count).add(4, rs.getString("LocationName"));
}

for(int i = 0; i<itemsList.size(); i++){ //delete duplicate locations

    int j = i+1;
    while (j<itemsList.size()){
        if (itemsList.get(i).get(0).equals(itemsList.get(j).get(0))){
            String newLocationList = itemsList.get(i).get(4) + ", " + itemsList.get(j).get(4);
            itemsList.get(i).set(4, newLocationList);
            itemsList.remove(j);
        }else{
            j++;
        }
    }
}
```

This allows items with multiple locations to be viewed as the same item. The arrayList holds the initial query results and the locations are added together in a string if they have the same ItemID. An arraylist is dynamic and can easily be modified/elements deleted.

These new locations appear on the jTable as shown below:

Location of purchases joined together into a long string when there are duplicates



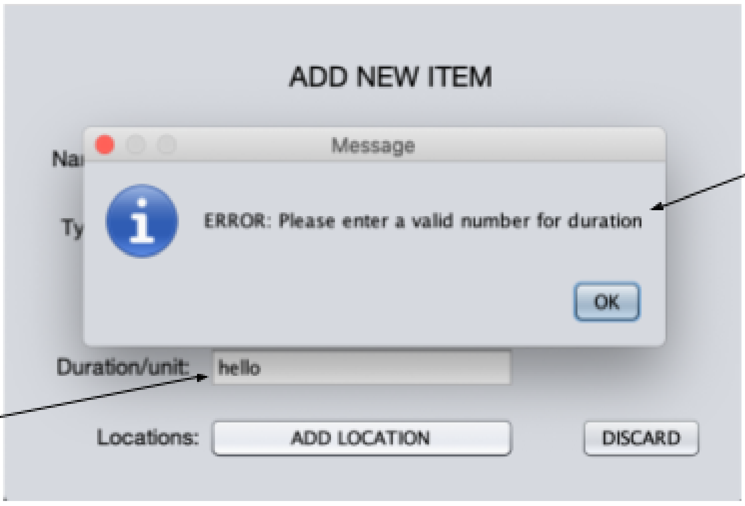
ID	ITEMS	TYPE	STOCK	LOCATION OF PURCHASE
50	Apple	Fruits	2	Sheng Siong, Mustafa
48	Banana	Fruits	0	Cold Storage, Little India, NTUC
55	Grapes	Fruits	2	Wholefood, null, Mustafa
64	Mango	Fruits	1	Wholefood, Sheng Siong, Cold Storage

This helps meet CFS 1, 2, and 6

Ingenuity:
The item information is not shown as it is stored in the database (with duplicates). Instead, it is presented cleanly with all the information of an item in one row.

7. Validation of all inputs

Validations were used on all inputs. This is how the validation would present to the user:



ERROR: Please enter a valid number for duration

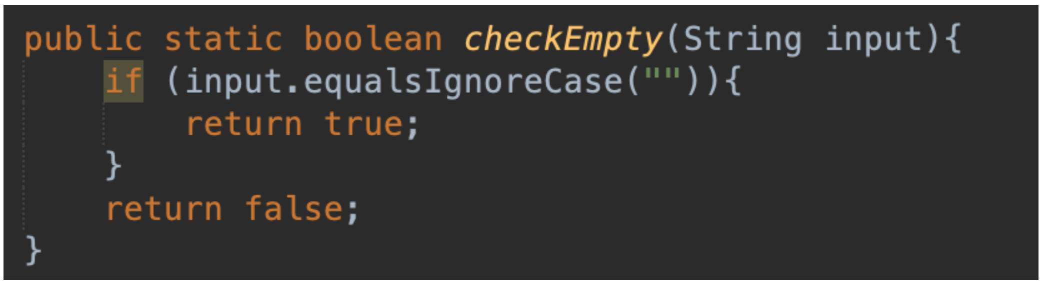
hello

The error message is displayed to the user telling the user what to change specifically

invalid input entered by the client

The following checks used were:

1. **Presence check**

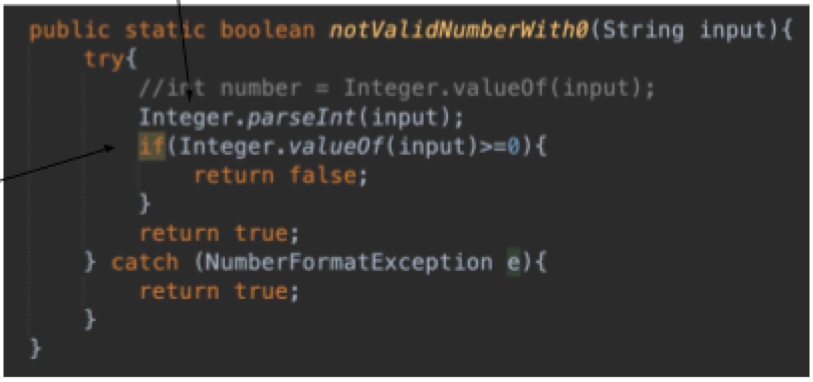


```java
public static boolean checkEmpty(String input){
    if (input.equalsIgnoreCase("")){
        return true;
    }
    return false;
}
```

The presence check is done to see if anything has been written in the input.

2. Range and type check

checks to see if the input is an integer or not → if it can't be parsed, then it throws an exception



```java
public static boolean notValidNumberWith0(String input){
    try{
        //int number = Integer.valueOf(input);
        Integer.parseInt(input);
        if(Integer.valueOf(input)>=0){
            return false;
        }
        return true;
    } catch (NumberFormatException e){
        return true;
    }
}
```

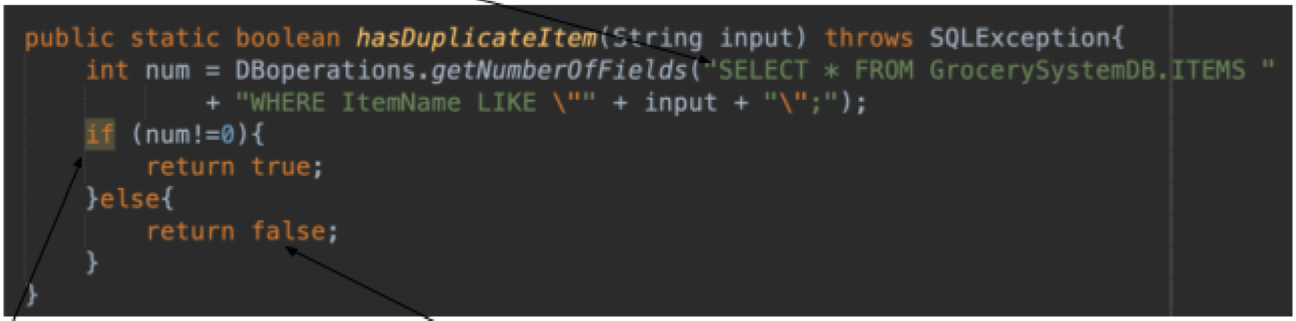
checking to see if the input fits in the required range

A similar validation exists for numbers that are not inclusive of 0 (numbers greater than or equal to 1), since there are two different ranges depending on the item property.

The type check uses feature 5 for implementation.

3. Duplicate entry validation

Query that returns the number of items with a certain name



```java
public static boolean hasDuplicateItem(String input) throws SQLException{
    int num = DBoperations.getNumberOfFields("SELECT * FROM GrocerySystemDB.ITEMS "
            + "WHERE ItemName LIKE \"" + input + "\";");
    if (num!=0){
        return true;
    }else{
        return false;
    }
}
```

If not 0 results are returned then that indicates a duplicate and the program returns true

If the query has no results, that means there are no items in the database with that name and the program returns false

A similar validation was made for checking duplicate locations.

Ingenuity:
The validation errors are easy for the user to understand what to fix/change.

8. Additional libraries

Additional java libraries are used for the functionality of the product.

```
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;

import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
```

```
import java.time.Month;
import java.time.LocalDate;
```

The java.sql package allows for connection with the database, allowing feature 1 to work. The javax.swing package is required for the creation of the GUI and the java.time package is needed for finding the user's current date.

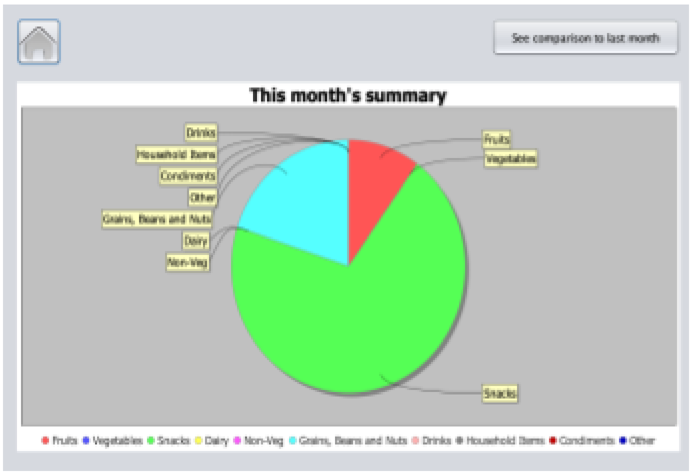
An external library, JFreeChart, is used to create the visual display of the monthly expenditure summary and comparison using charts.

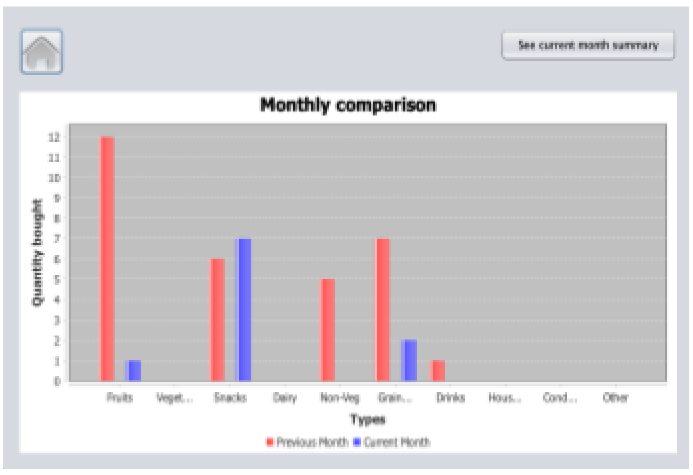
```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;
```

Using the jFreeChart class from the imported package.

```
JFreeChart chart = ChartFactory.createPieChart("This month's summary", dataset);

ChartPanel CP = new ChartPanel(chart);
CP.setSize(767, 394);
pieChartPanel.add(CP);
```

This results in the front end looking like this:


See comparison to last month
This month's summary
Drinks
Household Items
Condiments
Other
Grains, Beans and Nuts
Dairy
Non-Veg
Fruits
Vegetables
Snacks
● Fruits ● Vegetables ● Snacks ● Dairy ● Non-Veg ● Grains, Beans and Nuts ● Drinks ● Household Items ● Condiments ● Other


See current month summary
Monthly comparison
Quantity bought
0 1 2 3 4 5 6 7 8 9 10 11 12
Fruits Veget... Snacks Dairy Non-Veg Grain... Drinks Hous... Cond... Other
Types
■ Previous Month ■ Current Month

jFreeChart helped meet CFS 10 and 11.

Ingenuity:
The summary is presented in a more visual way for the client, instead of having to see information as another table, making it easier for the user to process that information as a visual display.

Sources

For coding:

- Stack Overflow
- Youtube
- GeeksforGeeks
- https://docs.oracle.com/javase/tutorial/uiswing/components/table.html

Images used:

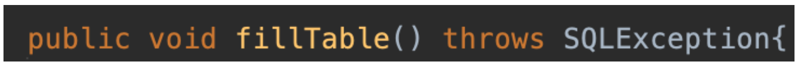
- Assorted Jars on Blue Shelf Cabinets by freestocks.org on Pexels (royalty-free image)

Extensibility

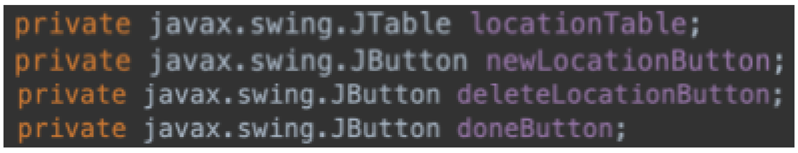
1. Good variable and method names

Variable, method and swing component names are clear in their function and purpose, making the code easier to understand. For further development, little effort needs to be spent understanding what each thing does.

Examples:



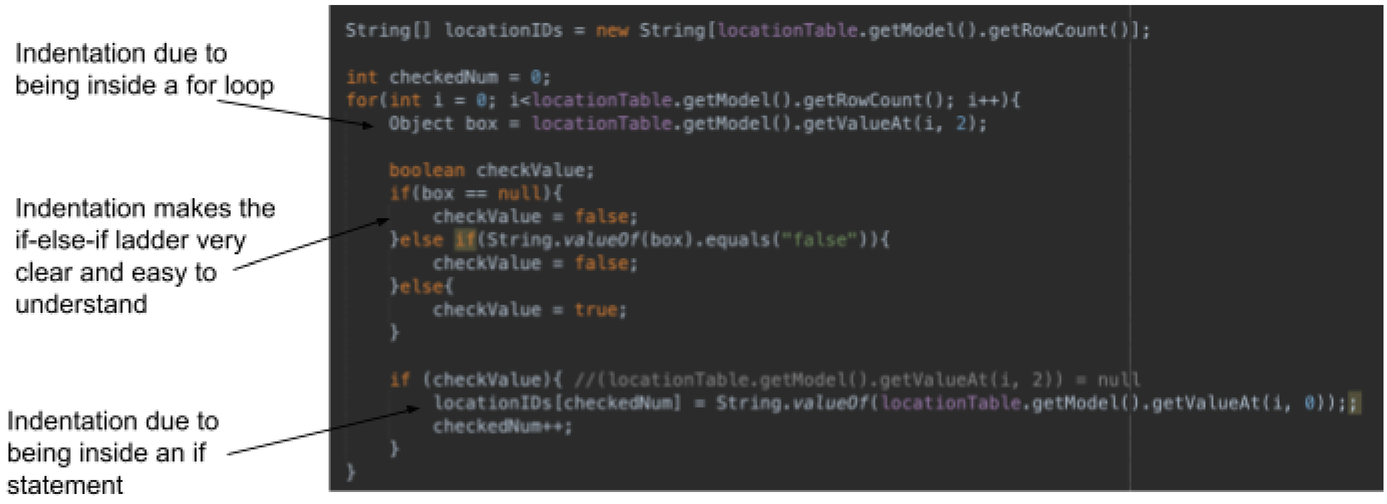
```
public void fillTable() throws SQLException{
```



```
private javax.swing.JTable locationTable;
private javax.swing.JButton newLocationButton;
private javax.swing.JButton deleteLocationButton;
private javax.swing.JButton doneButton;
```

2. Indentation

Indentation was used throughout the program to make the code easier to read.


Indentation due to being inside a for loop
Indentation makes the if-else-if ladder very clear and easy to understand
Indentation due to being inside an if statement

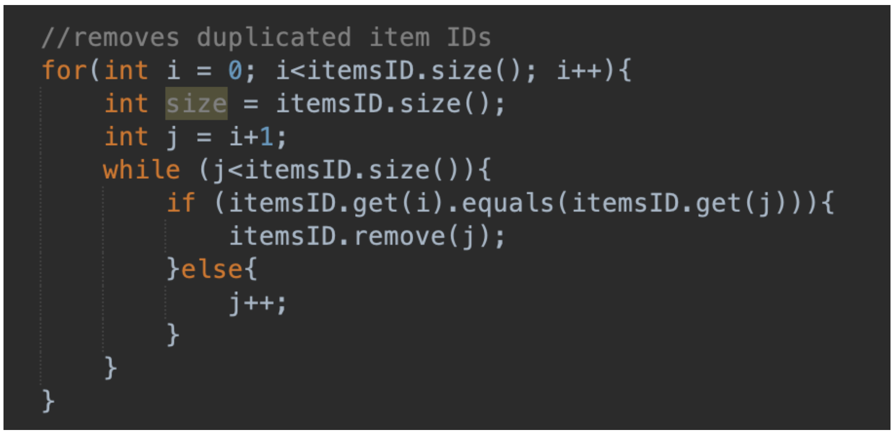
```java
String[] locationIDs = new String[locationTable.getModel().getRowCount()];

int checkedNum = 0;
for(int i = 0; i<locationTable.getModel().getRowCount(); i++){
    Object box = locationTable.getModel().getValueAt(i, 2);

    boolean checkValue;
    if(box == null){
        checkValue = false;
    }else if(String.valueOf(box).equals("false")){
        checkValue = false;
    }else{
        checkValue = true;
    }

    if (checkValue){ //(locationTable.getModel().getValueAt(i, 2)) = null
        locationIDs[checkedNum] = String.valueOf(locationTable.getModel().getValueAt(i, 0));;
        checkedNum++;
    }
}
```

This helps with extensibility since the program is organised and easy to understand.

3. Use of comments

Comments are used in the program to indicate what a certain piece of code does. The comments help break up the code into smaller parts, making it more straightforward to understand.

An example:

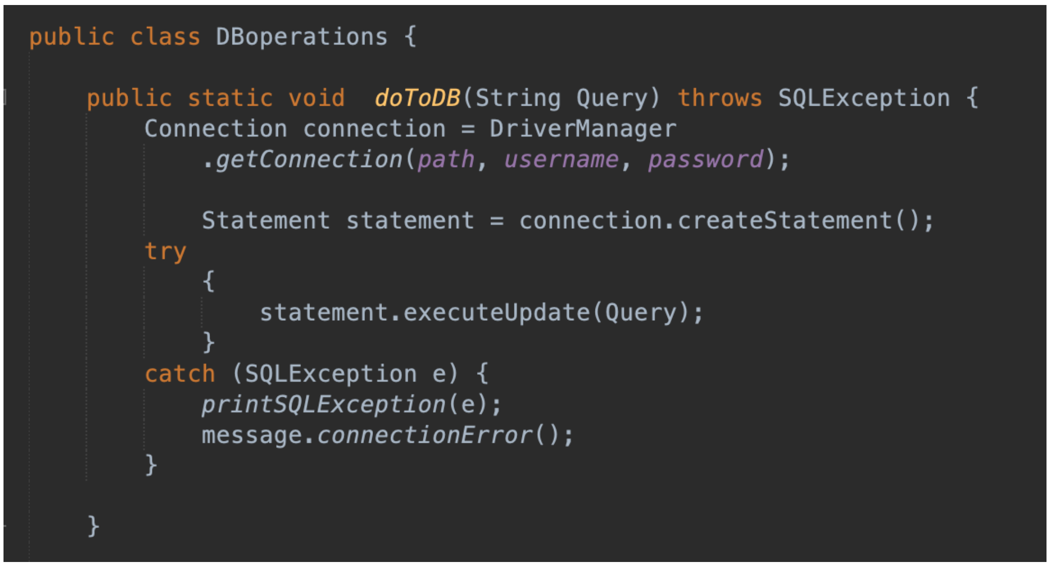


```java
//removes duplicated item IDs
for(int i = 0; i<itemsID.size(); i++){
    int size = itemsID.size();
    int j = i+1;
    while (j<itemsID.size()){
        if (itemsID.get(i).equals(itemsID.get(j))){
            itemsID.remove(j);
        }else{
            j++;
        }
    }
}
```

4. Methods hold sub-programs that are frequently used

Some pieces of code are used very frequently in the whole program. These methods can be called anywhere in the program instead of having to rewrite the whole code over again

Since database operations are used frequently, the DBoperation methods increase extensibility.



```
public class DBoperations {

    public static void  doToDB(String Query) throws SQLException {
        Connection connection = DriverManager
            .getConnection(path, username, password);

            Statement statement = connection.createStatement();
        try
            {
                statement.executeUpdate(Query);
            }
        catch (SQLException e) {
            printSQLException(e);
            message.connectionError();
        }

    }
```

There are similar methods for validation checks.

Word count: 977