

CSE3OAD/CSE4OAD – Assignment 1

Due Date: 11 September 2020 (11:59 PM)

Assessment: This assignment 1 is worth 30% of the final mark for CSE3OAD and CSE4OAD.


This is an individual assignment.

Copying, Plagiarism: Plagiarism is the submission of somebody else's work in a manner that gives the impression that the work is your own. The Department of Computer Science and Information Technology treats plagiarism very seriously. When it is detected, penalties are strictly imposed. Students are referred to the Department of Computer Science and Information Technology's Handbook and policy documents with regard to plagiarism.

No extensions will be given: Penalties are applied to late assignments (5% of your total assignment mark given is deducted per day, accepted up to 5 days after the due date only). If there are circumstances that prevent the assignment being submitted on time, an application for special consideration may be made. See the departmental Student Handbook for details. Note that delays caused by computer downtime cannot be accepted as a valid reason for a late submission without penalty. Students must plan their work to allow for both scheduled and unscheduled downtime. Assignments submitted more than 5 days late (i.e. after 18 September 2020) will receive the mark of 0.

Return of Assignments: Students are referred to the departmental Student Handbook for details.

What is in my Fridge



Id	Item	QTY	Section	Bought
6	Paddle Pop	1	FREEZER	15 days ago
19	Fish	1	MEAT	12 days ago
33	Beef	3	MEAT	8 days ago

ADD UPDATE ONE DELETE EXIT

Introduction

The aim of the assignment is to create an application to maintain a collection of groceries stored in your fridge. The groceries and a list of items are stored in a MySQL database.

A SQL script is provided to create the tables in the mysql database. Note:

- 2 tables will be created upon execution of the script: a grocery table and an item table
- you should change the name of the database in the script if you are using the MySQL database on latcs7. In that case, the name of the database will be your username not fridge.
- Sample records will also be added to these 2 tables

The application consists of the following classes:

- Grocery
- Item
- FridgeDSC
- FridgeFX

These classes are either provided as complete java files or as templates that you must complete. A css file is also provided.

Grocery

The Grocery class represents a grocery item bought and stored in your fridge. You do not have to add anything to this class, the complete class code is provided. As can be seen from the provided code, the class has the following attributes:

- **id**: the unique identifier of a grocery.
- **item**: an instance of the **Item** class.
- **Item** class, (provided) has the following attributes:
 - **name**: the name of an item from which you can pick to make a Grocery object
 - **expires**: whether or not this item can expire
- **date**: the date the grocery item was purchased/added to the fridge.
- **quantity**: quantity of such grocery item bought/added to the fridge at the same time.
- **section**: which section of the fridge is the grocery stored.

The **id** attribute of a grocery is of type **int** and is auto generated by the database, as can be seen in the SQL script. The Grocery class has a main method so that you can compile it on its own and test to make sure that it works. See the code.

Item

The second class, **Item**, has a one to one mapping with the **Grocery** class where **Grocery** class has one (and only one) instance of **Item**. You do not have to add anything to this class, the complete class code is provided. As can be seen from the provided code, the class has following attributes:

- **name**: the unique identifier for an Item
- **expires**: a Boolean specifying if an Item has an expiration date or not. The default is False.

The Item class also has a main method for testing.

FridgeDSC

The third class, FridgeDSC, is the data source controller. A skeleton of this class is provided.

You should use this class to create methods to send SQL commands to database. You will send and receive data.

FridgeFX

The fourth class, FridgeFX, provides the graphical user interface for the users to interact with the system. This class is implemented in JavaFX. A skeleton for this class is provided.

You should use this class to create methods to create the GUI aspects of the system.

Task 1

Implement the **FridgeDSC** class. A skeleton of the class is provided, which indicates the methods that you are required to implement. Note the following database management methods

DriverManager
Connection
Statement
PreparedStatement

There are method stubs that you will need to connect to a database/table and perform the required task – the SQL queries (**String** variable) for each of the method will be provided in the code.

NOTE: Carefully read the comments provided throughout the skeleton class – they provide useful hints on how to proceed with each method stubs/tasks. You are required to implement each part labelled with

/ TODO 1-xx - TO COMPLETE ******

Task 2

Implement the **FridgeFX** class. When the system is started, a screen similar to the one shown in Figure 1 is displayed. Your **FridgeFX** class should create an instance of **FridgeDSC** class and use that object instance methods to perform the *create, read, update, delete (CRUD)* operations described below;

NOTE: Carefully read the comments provided throughout the skeleton class – they provide useful hints on how to proceed with each method stubs/tasks. You are required to implement each part labelled with

/ TODO 2-xx - TO COMPLETE ******

Describing the User Interface Flow/Requirements

Figure 1.1 shows the main interactive controls at the top of the application.

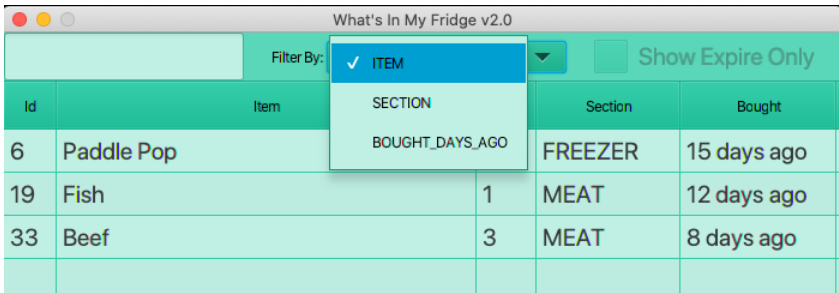


Figure 1.1 – “Filter By:” ChoiceBox in action

- A **TextField** – this allows the user to filter the **TableView** data
- A **ChoiceBox** – this allows the user to select which **Column** of the **TableView** to use as filter target
 - When ITEM option (default selection) is selected, the **TextField** filter will target the values in the **TableView** “Item” column.
 - When SECTION option is selected, the **TextField** filter will target the values in the **TableView** “Section” column.
 - When BOUGHT_DAYS_AGO is selected, it does the following:
 - Enables the “Show Expire Only” **CheckBox**
 - Sets the **TextField** filter to the **TableView** “Bought” column (only it’s numerical value), listing all groceries bought on the filtered value days **and prior**
- A **CheckBox** – this is currently disabled; this control is enabled when the “Filter By:” BOUGHT_DAYS_AGO option is selected.

NOTE: the **Grocery** class has an attribute of type **Item** class (both classes are provided to you); Item class has a boolean expires attribute; The “Show Expire Only” Checkbox, when selected, will also filter out those groceries elements having an item with attribute expires set to true;

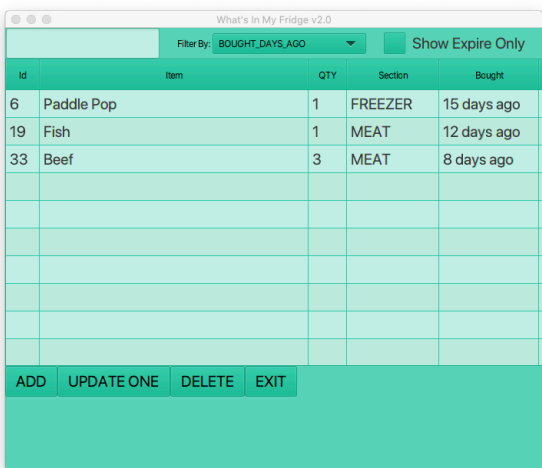


Figure 1.2 – BOUGHT_DAYS AGO filter, “Show Expire Only” **not** selected. Paddle Pops don't Expire

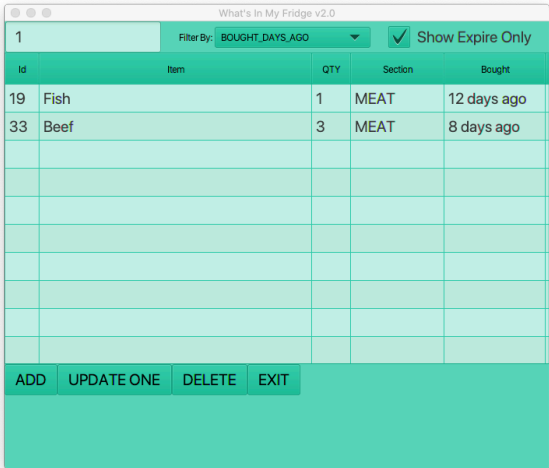


Figure 1.3 – BOUGHT_DAYS AGO filter, “Show Expire Only” selected. Paddle Pops are now not displayed.

The next control is the **TableView**.

The **TableView** displays the groceries in the collection, one grocery per row. Each row of the **TableView** is selectable; In order to use the “UPDATE ONE” or the “DELETE” button, the relevant **TableView** row (a grocery) must be selected by the user;

Adding a new grocery:

Clicking the “ADD” button reveals a hidden container (see Figure 2)

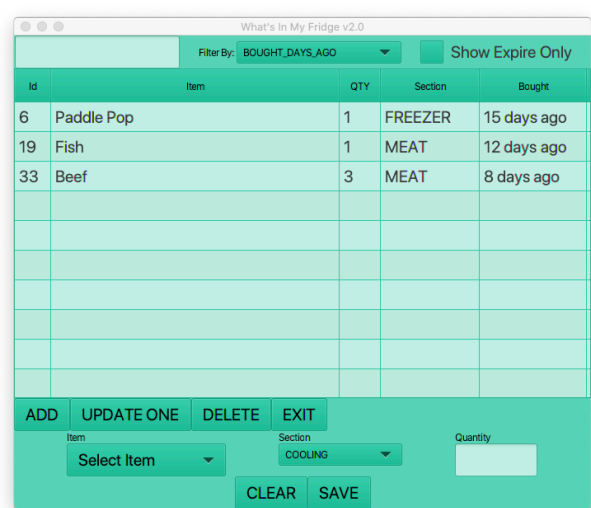


Figure 2 – Adding a grocery

You are then provided with 3 controls:

- A **ComboBox** – lists all **Items** available by item name.
- A **ChoiceBox** – listing the possible sections in the fridge; the section values are defined as an **enum** in **FridgeDSC** class. (See Figure 2.2). Look at the code in **FridgeDSC.java**
- A **TextField** – user input for quantity of selected item the user is about to add to the fridge.

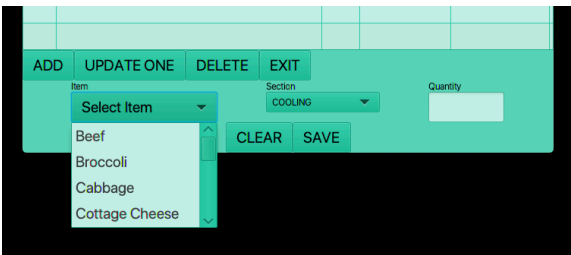


Figure 2.1 – the “Item” **ComboBox** listing

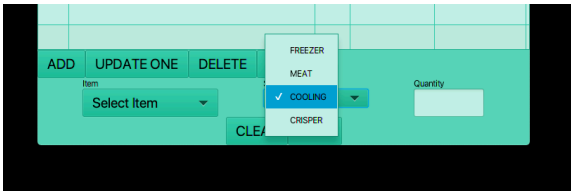


Figure 2.2 – the “Section” **ChoiceBox** listing

After selecting/entering some grocery information in the add controls, click the SAVE button, to create a new grocery entry (see Figure 2.3 and 2.4)

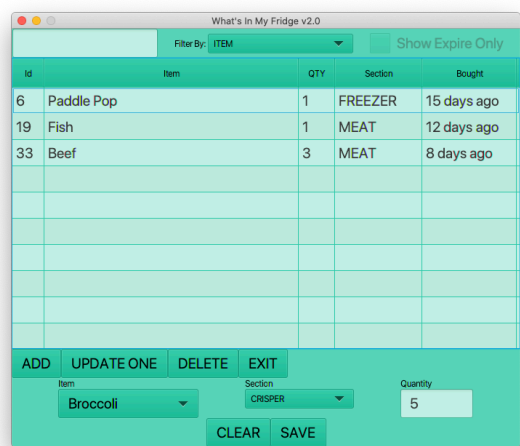


Figure 2.3 – entering new grocery information

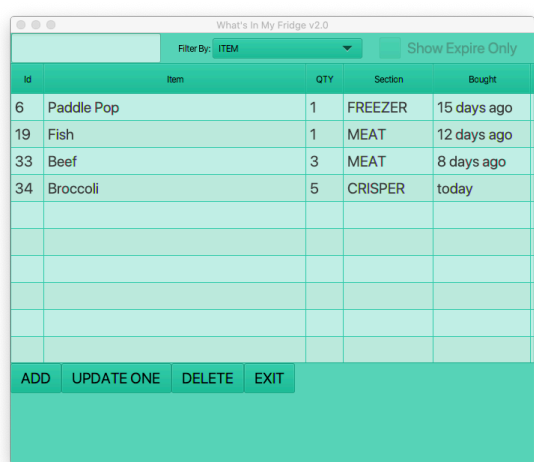


Figure 2.4 – saved new grocery, listed in the **TableView**

Update One grocery

The UPDATE ONE button decreases the quantity of a selected grocery in the TableView by one. If the selected grocery is already one it will prompt a relevant error message.

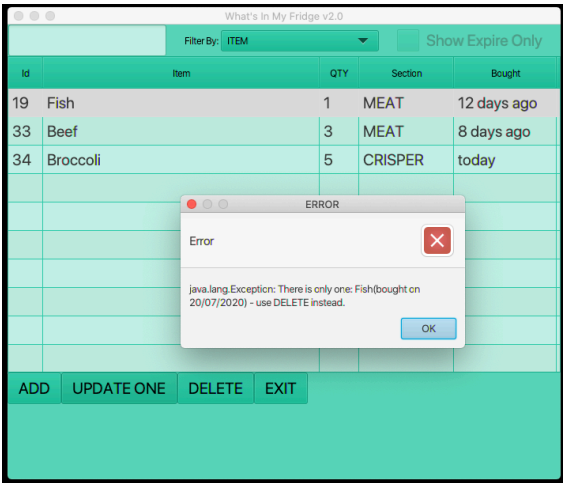


Figure 2.5 – try to UPDATE ONE grocery with quantity = 1

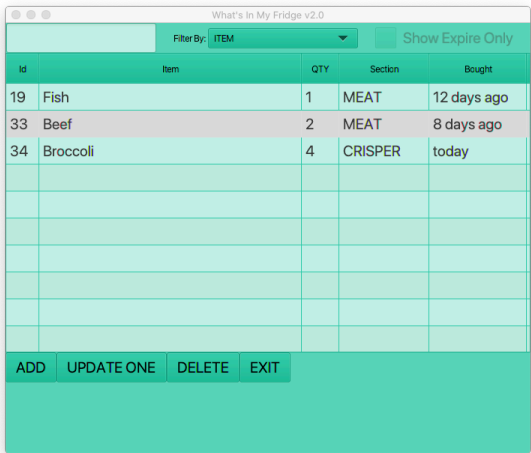


Figure 2.4 – successfully UPDATE ONE action on grocery (id: 33) with quantity = 3, now quantity = 2

Delete (one) Grocery

The DELETE button prompts user with a confirmation, and if user accepts (clicks OK button) deletes the selected grocery.

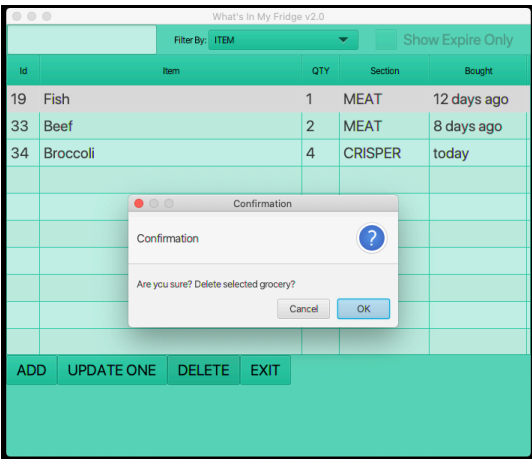


Figure 2.5 – DELETE selected grocery, user prompted for confirmation

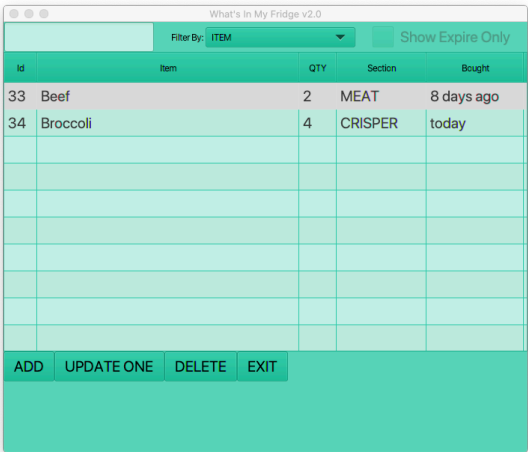


Figure 2.6 – successful DELETE of grocery (id: 19) confirmation

Exit from the application

The Exit button prompts user with a confirmation, and if user accepts (clicks OK button) exits the application. The groceries seen in the application should be synced to the mysql database so that when the application is started again the grocery state is maintained. Note that this could be happening all the time

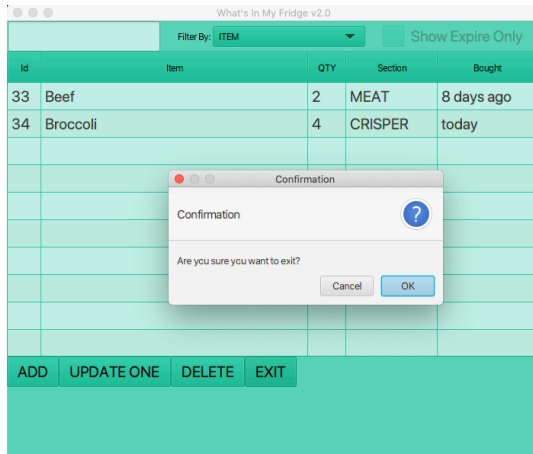


Figure 2.7 – EXIT user prompted for confirmation

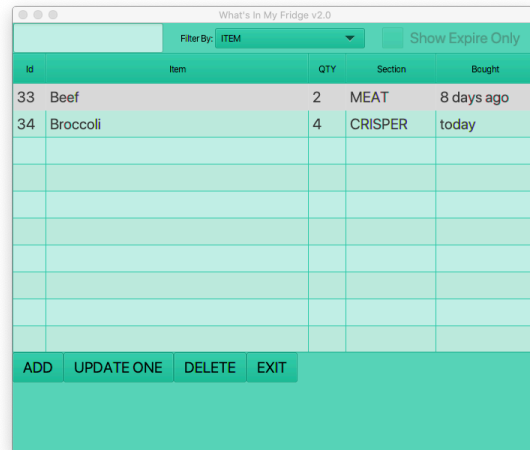


Figure 2.8 – The application when restarted

Use adequate means of alerting users of relevant events

Make sure you add a notification mechanism (example: Alert boxes) to Prompt user before executing a critical action (an add, an update or a delete) Inform user when an error has occurred; see Exception Handling for more requirements on error handling

Exception Handling

Make sure you add the exception handling code so that whenever an exception occurs, the program displays an alert which shows a brief message about the exception.

Implementation Suggestion

There are two approaches.

1. Start with the FridgeDSC.java, this has all the required interaction with the database, and includes all methods needed for the UI class (**FridgeFX**). Examine the main method in your **FridgeDSC** class and test all the needed required methods; see your lab sample solutions for some examples. Then Work on the FridgeFX.java. The disadvantage of this approach is that you need to wait until the lectures/labs on SQL are done.
2. In the JavaFX GUI class (**FridgeFX**), there is a commented-out section that includes the code for creating a list of groceries. If you use this commented out section for the TableView you can get the control and container elements done. Add Lambda function code stubs for the Buttons **setOnAction** methods and add the relevant Alert boxes. Then later when you have done the FridgeDSC.java you can replace the hard coded grocery list with the list that comes back from the DSC. The advantage of this is that you can start earlier, and don't have to wait for the SQL lectures.

Assignment 1 is based on topics covered in weeks 1 to 5.

You are strongly advised to regularly check the Assignment 1 LMS section for any hints, clarification or corrections regarding Assignment 1.

What to submit

Zip up all the java files and submit them as one file to the LMS.

All of your classes must be able to be compiled from their current directory. This means, they must not be contained in any package or project structure.

As for the database, you can use the one on latcs7 or on your local machine. The only requirement is that your program should work on the database tables **grocery** and **item** which must have the same structure as the one in the provided MySQL script (**CreateDatabaseScript.sql**).

For each class that you submit, you must include, as part of the comments, at the beginning of each file,

your name
your student ID
the subject code (CSE3OAD/CSE4OAD)

Can I make a better Application?

Yes. In fact, once you have completed the application as described here, you are encouraged to make it better. 10 marks are allocated as BONUS marks. These will be added to the overall marks. The overall mark will be capped at 100. Submit the better class as FridgeFX2.java. In the comments at the top of the code explain why your app is better. Don't spend too much time on this.

Mark Allocation

Task 1 –Completing the data source controller(FridgeDSC) 35 marks

find item (searchItem)	2
find grocery (searchGrocery)	4
find all items (getAllItems)	2
Find all groceries (getAllGroceries)	4
add (addGrocery)	9
update (useGrocery)	7
delete (removeGrocery)	7

Task 2 – Completing the JavaFX app (FridgeFX) - 60 marks

The data is obtained from the FridgeDSC and loaded into the appropriate array	4
The correct 5 columns defined	4
The data is loaded into the application correctly. This includes the TableView, combo box and choice box	4
The columns have the right names and are in the right order.	2
The right CSS styles, as shown in this document, should be applied. Note the supplied css file.	2
Sorting by Columns by clicking on column's header (N.B. be careful of order. BOUGHT_DAYS_AGO should be numerical order)	4
"filtered by" ITEM	4
"filtered by" SECTION	4
"filtered by" BOUGHT_DAYS_AGO (numerical filter)	4
"Show Expire Only" Check box	2
Clicking the ADD button makes the addBox visible	2
Clicking the SAVE button saves the details to both the TableView and the database. i.e adding works, and hides the addBox.	8
Clicking the CLEAR button resets the ADD fields, does not affect the database or the TableView array, and hides the addBox	3
Clicking UPDATE ONE reduces the quantity by one unless there is only one remaining. Then an exception is raised. Otherwise no confirmation is required. Both the quantity shown in the table and the database should be changed.	5
Clicking DELETE raises an Alert and on confirmation, removes the grocery. Both in the table and the database.	5
Clicking EXIT raises an Alert and on confirmation, exits the application.	3

Coding Standards, Proper naming of variables, indentation	5
--	----------

Total 100

Bonus marks for refactor for improved behaviour (capped at 100)	10
--	-----------