CSE3OAD/CSE4OAD – Assignment 1

Due Date: 3 PM, Friday September 14th, 2018

Assessment: This assignment 1 is worth 15% 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 3 PM, Friday September 21st, 2018) will receive the mark of 0.

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

Objectives: To design and implement an application in JavaFX and to use JDBC for data persistence.

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.

The system consists of the following main classes:

- **`Grocery`**
- **`Item`**
- **`FridgeDSC`**
- **`FridgeFX`**

The **`Grocery`** class represents a grocery item bought and stored in your fridge. 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 **`Item`** class.
    - **`Item`** class, (complete code also 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 actually automatically generated by the database, as can be seen in the SQL script.

(So you may ask: why do we have the constructor in the **`Grocery`** class? This will be left for you to figure out.)

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`**.

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

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

An SQL script is also provided to create the tables.

Note:

- 2 tables will be created upon execution of the script: a **grocery** table and an **item** table.
- Sample records will also be added to these 2 tables
- For this assignment, you will only query data from **item** table, that is, only select statements.

Task 1

Implement the **`FridgeDSC`** class. A skeleton of the class is provided, which indicates the methods that you are required to implement. Relevant database connectivity measure must be coded in (**`DriverManager`**, **`Connection`**, **`Statement`**, **`PreparedStatement`**) and the implementation for methods stubs require you to connect to a database/table and perform the required task – the 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 labeled with

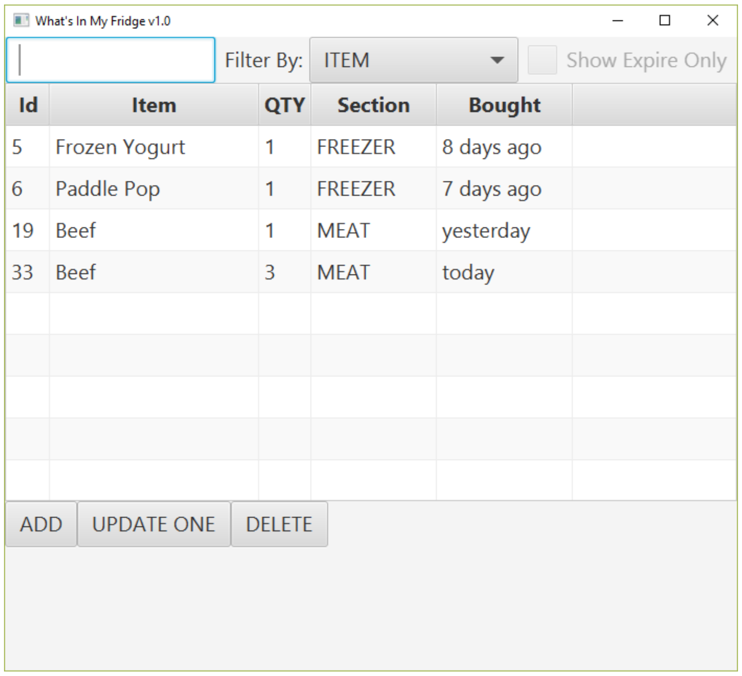
```
/* TODO 1-xx - TO COMPLETE ****************************************
```

Task 2

Implement the **`FridgeFX`** class. When the system is started, a screen similar to the one shown in Figure 1 is displayed. Your **`FrigdeFX`** 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 labeled with

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



Id	Item	QTY	Section	Bought
5	Frozen Yogurt	1	FREEZER	8 days ago
6	Paddle Pop	1	FREEZER	7 days ago
19	Beef	1	MEAT	yesterday
33	Beef	3	MEAT	today

Figure 1 - Screen displayed by **`FridgeFX`**

Describing the User Interface Flow/Requirements

Figure 1 shows the overall User Interface of the application you are required to build; Let us discuss each section;

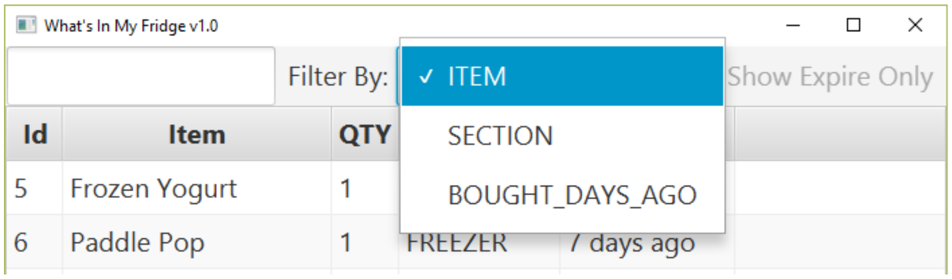
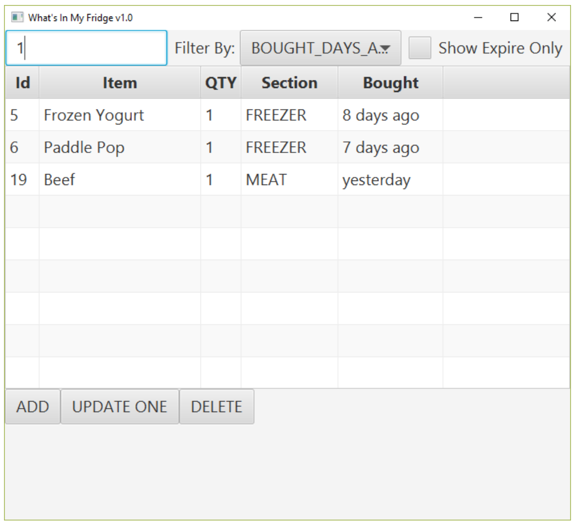


Figure 1.1 – *"Filter By:"* **ChoiceBox** in action

Figure 1.1 show the following interactive controls:

- 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.

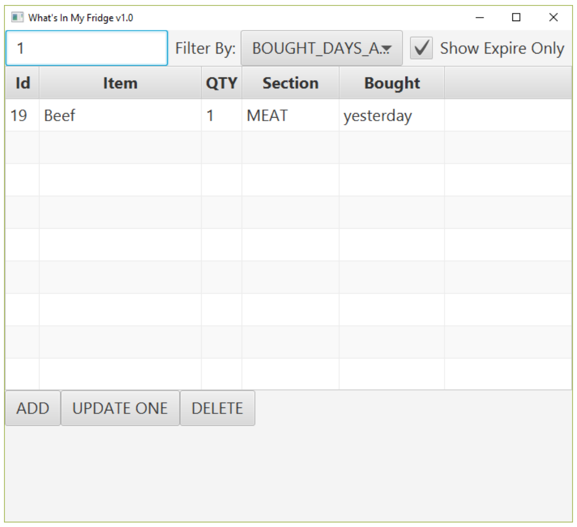
NOTE: the **`Grocery`** class has an attribute of type **`Item`** class (both classes are provided to you); **`Item`** class has a **`boolean expires`** attribute; The *"Show Expire Only"* Checkbox, when selected, will also filter out those groceries elements having an item with attribute **`expires`**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



1

Id	Item	QTY	Section	Bought
5	Frozen Yogurt	1	FREEZER	8 days ago
6	Paddle Pop	1	FREEZER	7 days ago
19	Beef	1	MEAT	yesterday

Figure 1.2 – BOUGHT_DAYS AGO filter, *"Show Expire Only"* **not** selected



1

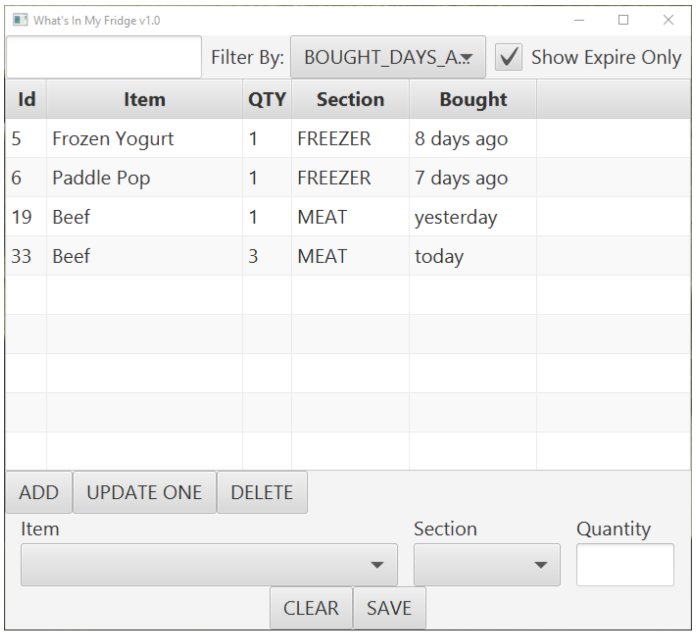
Id	Item	QTY	Section	Bought
19	Beef	1	MEAT	yesterday

Figure 1.3 – BOUGHT_DAYS AGO filter, *"Show Expire Only"* selected

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)

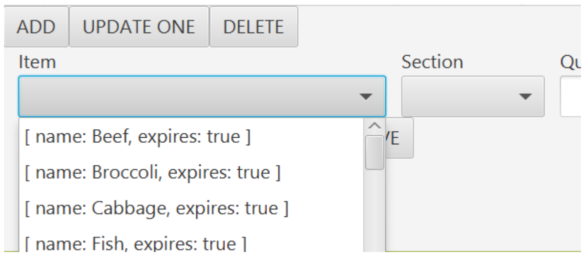


Id	Item	QTY	Section	Bought
5	Frozen Yogurt	1	FREEZER	8 days ago
6	Paddle Pop	1	FREEZER	7 days ago
19	Beef	1	MEAT	yesterday
33	Beef	3	MEAT	today

Figure 2 – Adding a grocery

You are then provided with 3 controls:

- A **ComboBox** – lists all items stored – each item is an instance of Item class; For simplicity, we are displaying the **`toString()`** result of each item instance as value of the **ComboBox** list. (see Figure 2.1)
- A **ChoiceBox** – listing the possible sections in the fridge; the section values is defined as an **`enum`** in **`FridgeDSC`** class. (See Figure 2.2)
- A **TextField** – user input for quantity of selected item the user is about to add to the fridge.



```
[ name: Beef, expires: true ]
[ name: Broccoli, expires: true ]
[ name: Cabbage, expires: true ]
[ name: Fish, expires: true ]
```

Figure 2.1 – the *"Item"* **ComboBox** listing

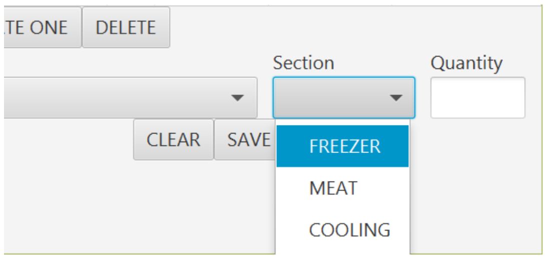
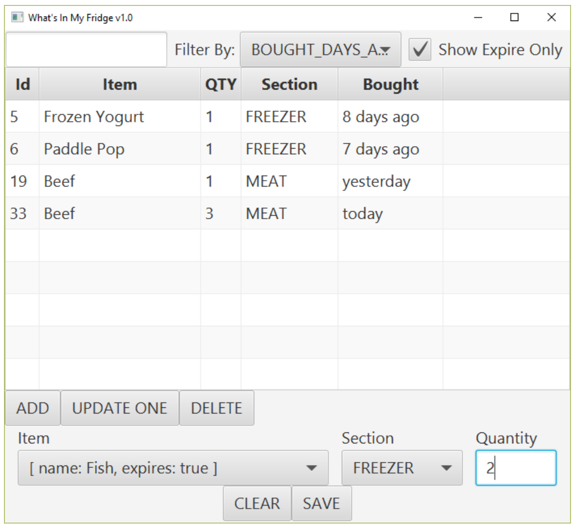


Figure 2.2 – the *"Section"* **ChoiceBox** listin

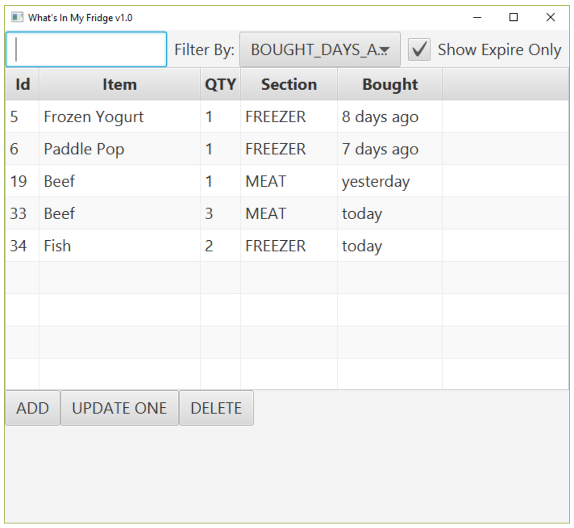
After selecting/entering some grocery information in the add controls, click the SAVE button, to create a new grocery entry (see Figure 2.3 and 2.4)



Id	Item	QTY	Section	Bought
5	Frozen Yogurt	1	FREEZER	8 days ago
6	Paddle Pop	1	FREEZER	7 days ago
19	Beef	1	MEAT	yesterday
33	Beef	3	MEAT	today

Item	Section	Quantity
[name: Fish, expires: true]	FREEZER	2

Figure 2.3 – entering new grocery information

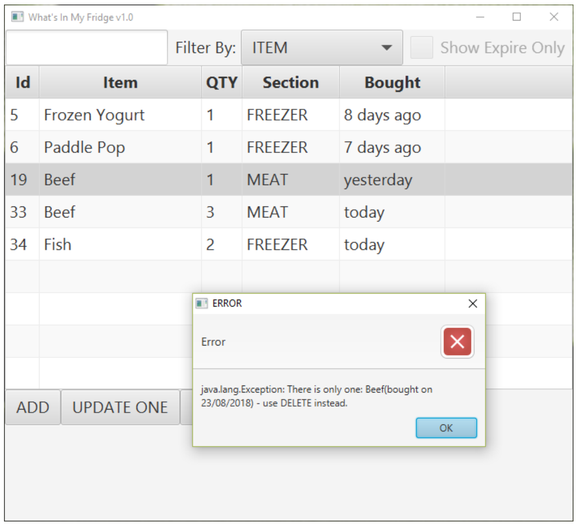


Id	Item	QTY	Section	Bought
5	Frozen Yogurt	1	FREEZER	8 days ago
6	Paddle Pop	1	FREEZER	7 days ago
19	Beef	1	MEAT	yesterday
33	Beef	3	MEAT	today
34	Fish	2	FREEZER	today

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. If the selected grocery is already one it will prompt a relevant error message.



What's In My Fridge v1.0

Filter By: ITEM ☐ Show Expire Only

Id	Item	QTY	Section	Bought
5	Frozen Yogurt	1	FREEZER	8 days ago
6	Paddle Pop	1	FREEZER	7 days ago
19	Beef	1	MEAT	yesterday
33	Beef	3	MEAT	today
34	Fish	2	FREEZER	today

ADD UPDATE ONE

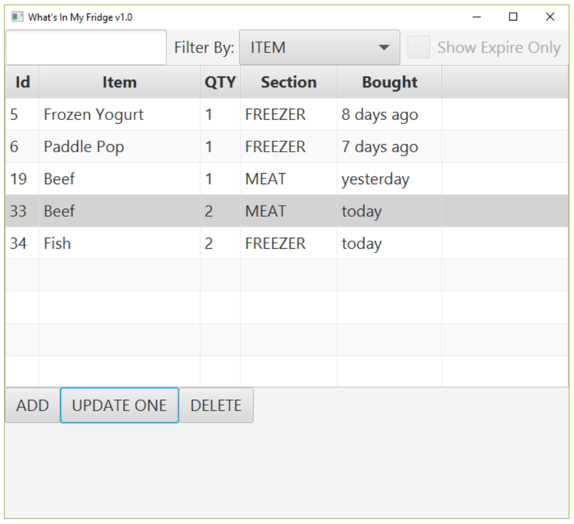
ERROR

Error

java.lang.Exception: There is only one: Beef(bought on 23/08/2018) - use DELETE instead.

OK

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



What's In My Fridge v1.0

Filter By: ITEM ☐ Show Expire Only

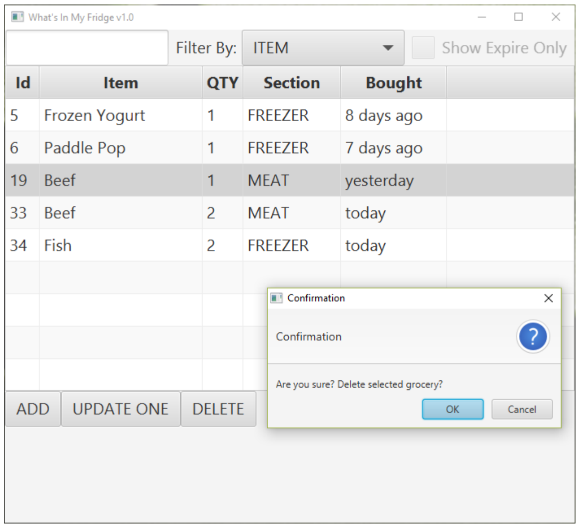
Id	Item	QTY	Section	Bought
5	Frozen Yogurt	1	FREEZER	8 days ago
6	Paddle Pop	1	FREEZER	7 days ago
19	Beef	1	MEAT	yesterday
33	Beef	2	MEAT	today
34	Fish	2	FREEZER	today

ADD UPDATE ONE DELETE

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.



What's In My Fridge v1.0

Filter By: ITEM ☐ Show Expire Only

Id	Item	QTY	Section	Bought
5	Frozen Yogurt	1	FREEZER	8 days ago
6	Paddle Pop	1	FREEZER	7 days ago
19	Beef	1	MEAT	yesterday
33	Beef	2	MEAT	today
34	Fish	2	FREEZER	today

ADD UPDATE ONE DELETE

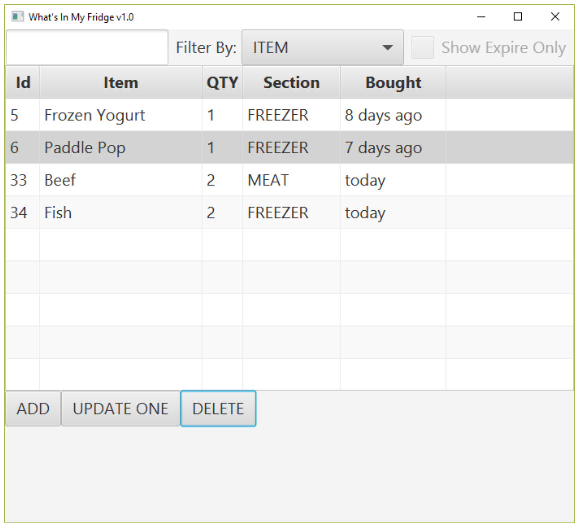
Confirmation

Confirmation

Are you sure? Delete selected grocery?

OK Cancel

Figure 2.5 – DELETE selected grocery, user prompted for confirmation



What's In My Fridge v1.0

Filter By: ITEM ☐ Show Expire Only

Id	Item	QTY	Section	Bought
5	Frozen Yogurt	1	FREEZER	8 days ago
6	Paddle Pop	1	FREEZER	7 days ago
33	Beef	2	MEAT	today
34	Fish	2	FREEZER	today

ADD UPDATE ONE DELETE

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

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 display an alert which shows a brief message about the exception.

Implementation Suggestion

You are encouraged to implement Assignment 1 in the following order:

1. In the JavaFX UI class (**`FridgeFX`**), code the control and container elements;
2. Code the data source controller (**`FridgeDSC)`**, all the required interaction with the database, the methods needed for the UI class (**`FridgeFX`**) to interact with this data source controller
    - Implement a main method in your **`FridgeDSC`** class to facilitate testing all the needed required methods; see your lab sample solution for some examples.
3. Go back to the UI class (**`FridgeFX`**)
    - add Lambda function code stubs for the Buttons **`setOnAction`** methods;
    - add the relevant Alert boxes
4. Test the overall system thoroughly, making sure it compiles on **`latcs`**
5. The above 4 points are suggestions that can help you start the assignment and finish it by the due date.
6. Assignment 1 is based on topics covered in weeks 1 to 5.

Consultation and other resources

1. Consultation sessions will be announced (LMS) prior to the submission date of this assignment 1.
2. You are strongly advised to regularly check the Assignment 1 LMS section for any hints, clarification or corrections regarding Assignment 1.

Files Provided

- `Item.java`
- `Grocery.java`
- `FridgeDSC.java`
- `FridgeFX.java`
- `CreateDatabaseScript.sql`

What to submit

Electronic copy of all of the classes required to run the application, including those that are provided, are to be submitted to latcs8 using the **`submit OAD <directory or filename>`** command

Note that the submission is not through 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.

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 (with surname in capital),
- your student ID,
- your user name, and
- the subject code (CSE3OAD or CSE4OAD)

Assignment submissions without any of these will have 5% of the mark deducted.