Curtin University
School of Electrical Engineering, Computing and Mathematical Sciences
Object Oriented Program Design (COMP1001)
Semester 1, 2018

Assignment

Due: ~~09:00, Monday, 28 May 2018~~
Week 14 (beginning Monday, 28 May 2018)
within the first 5 minutes of your scheduled prac
See section 4 – submission requirements.
Weight: 20% of the unit mark.

1 Logistics

This is not an assignment which can be put aside a few days before it is due. **If you leave working on the assignment until a week or so before it is due, then you will NOT be able to complete it successfully!** Your final mark will be directly coupled to:

- Your ability to communicate and to read and comprehend text based material.
- Your ability to demonstrate an understanding of how to design, implement and test an Object Oriented software application.
- Your ability to pro-actively seek assistance when required.
- Your ability to ensure that progress on the assignment is made frequently.

When attempting this assignment, please keep in mind that your goal is to refine your algorithm design and implementation skills, as well as deepening your understanding of Object Orientation.

2 The Task

Food can be classified into four distinct groups – Meat, Grain Products, Fruit, and Vegetables. Each of these groups have attributes in common, and attributes distinct to themselves.

Your task is to design a program that manages items of food, classifying them appropriately, and organising their storage accordingly. Your algorithm must handle errors elegantly as discussed in the lectures and practicals.

2.1 Food

Each type of food will be represented using a separate class.

2.1.1 Meat

Responsible for the following information:

- name:
- cut
- weight
- storage temperature
- useby date
- packaging

2.1.2 Grain

Responsible for the following information:

- name:
- type
- volume
- storage temperature
- best before date
- packaging

2.1.3 Fruit

Responsible for the following information:

- name:
- type
- number of pieces
- storage temperature
- useby date
- packaging

2.1.4 Vegetables

Responsible for the following information:

- name:
- weight
- storage temperature
- best before date
- packaging

2.2 Storage

The storage of food items will be divided into 3 areas: Freezer, Fridge and Pantry. For the purpose of this assignment, these will be represented by a 2-dimensional array, having 3 rows and some number of columns. Each storage location has a different temperature range. If a food product is outside of the given ranges, it is invalid.

2.2.1 Freezer

Stores food with a storage temperature in the range -27.0 to -5.0 degrees Celsius.

2.2.2 Fridge

Stores food with a storage temperature in the range -2.0 to 6.0 degrees Celsius.

2.2.3 Pantry

Stores food with a storage temperature in the range 8.0 to 25.0 degrees Celsius.

2.3 Menu

The program should begin by displaying a text-based menu with the following options:

- Add food
- Remove food
- Display contents
- Find expired
- Read in storage
- Write out storage
- Exit

Add food should add an item of food, with the values read from user input. If the storage has not yet been read in from file, or if the particular storage location is full, the program should output an appropriate error message and return to the menu, without adding the item.

Remove food should prompt use user for a storage area and location, and delete the item.

Find expired should cycle through all items in the storage facility looking for expired food. If food has expired, the program should output the details, including the location, to the user.

Display contents should output to the screen all the contents of a particular storage location – for example, the Fridge.

Read in storage should prompt the user for a file name and read in a (possibly empty) storage facility. The Storage facility should only be constructed once. Subsequent attempts to construct it should fail, and an appropriate error message output to the user. Refer to the File I/O section for more information.

Write out storage should prompt the user for a file name and write to this file a (possibly empty) storage facility.

Exit should gracefully exit the program. This must be the only way of exiting the program – no other function should allow the program to finish.

2.4 Validation

Both file input and user input should be pre-validated appropriately (validated before constructing the objects) where required.

The following values require validation:

- weight, volume: a positive real number between 0.2 and 5.0
- number of pieces: a positive integer between 1 and 20
- date – cannot be in the past

2.5 Classes

Your program should make good use of classes and submodules. In addition to the previously mentioned classes, others will be required. Students should give careful thought to the best arrangement of classes for the application. Note that the class that contains main is not permitted any class fields. You should have at least 7 classes.

2.6 Documentation

You must thoroughly document your code using block comments (/*…*/) and in-line comments (//…). For each Class, you must place a comment at the top of the file containing your name, the file name, the purpose of the class and any modification history. For each method you write you must place a comment immediately above it explaining its purpose, it's import and export values, and how it works. Other comments must be included as necessary.

3 Input/Output

Input and output should be done using the techniques from the lectures and practical sessions.

3.1 Input File

The input file will contain sufficient information to construct a storage facility; at a minimum it will contain the sizes of the Fridge, Freezer and Pantry. These will always be the first 3 lines in the file, but in no particular order. For example:

```
Pantry, 100
Freezer, 20
Fridge, 50
```

The input file could additionally have food items to be stored upon construction. Your program must decide whether it is valid, and in which storage location it belongs. For example:

```
Meat, Beef, fillet, 1.3, -18.0, 01/09/2018, Vacuum packed
Vegetable, carrot, 2.0, 4.0, 01/08/2018, bag
Fruit, apple, golden delicious, 6, 3.0, 01/07/2018, loose
Grain, Wheat, Weetbix, 2.0, 18.0, 01/10/2018, box
```

Note: All values on a line will be comma separated.

If any of the first 3 lines are invalid, the storage facility should not be constructed. If any of the food items are invalid, they should not be added, but the program should continue to process the remaining food items. If a storage area becomes full, no more items should be added to that particular area.

3.2 Output File

The output file must be written in exactly the same format as the input file.

3.3 User Input

All user input should be validated where appropriate.

3.4 User Output

All output to the user should be formatted in a pleasant, easy to read manner.

4 Submission Requirements

Electronic copies of all assignment documents must be kept in your Curtin user accounts, and not edited after the due date as the assignment will be tested in your tutorial sessions. These should be in a directory called OOPDAssignment, located immediately under your home directory. Failure to do this will result in 0 marks being allocated to the testing portion of the marking.

The assignment must be submitted in printed (not handwritten) form. They should be submitted to your tutor in your scheduled practical session. You must also submit your assignment through Blackboard for record keeping purposes (This submission will not usually be marked).

The assignment cover sheet must be attached to the front and the information specified must be supplied and correct. You MUST sign the declaration on the cover sheet. Failure to conform to these requirements will result in your assignment not being accepted and not being marked.

Each assignment submission should be made in hard copy form and should contain:

- A completed Cover Page (can be found on Blackboard)
- A complete set of pseudocode that describes the system you created
- A complete set of source code for the designed software. The Java code must be a valid implementation of your supplied pseudocode. Only submit .java files, not .class files.

IMPORTANT NOTE: Your text editors wordwrap is not to be relied on!!! It is suggested you use a word processor, using courier size 10 font.

Please note that assessments submitted in the Department of Computing must comply with the Department of Computing Coding Standard, which can also be found on Blackboard. All code must be fully documented. Late submissions will incur the penalties laid out in the Unit Outline. Anything after the due date is considered a late submission, even if only a minute late.

4.1 Assignment Document Ordering

Your hard copy assignment MUST be organised in the following order:

- Completed and filled in cover page
- Completed and filled in declaration
- Pseudo code for main, then Java for main
- Pseudo code then Java for each class, in the order that the classes are used.

The assignment must be stapled together at the top left corner, or bound, to make it easy to read.

5 Marking Strategy

The pseudocode algorithm and Java implementation will both be marked. Your java code will be assessed by its suitability as a valid implementation of your supplied pseudocode. You will be asked in the last practical session to demonstrate your program and answer questions on the design. If you cannot answer the questions satisfactorily, you will receive a mark of zero for the assignment. Marks will be allocated for the appropriate choices of classes, the overall functionality, and adhering to best programming practices as discussed in the lectures and practicals.

You cannot pass the assignment without attending your weekly tutorial and attempting your signoff questions. This assignment will be weight by the average of your signoff marks according to the table below:

Average

% Total	% Effect
77.5 - 100	100
67.5 – 77.4	90
57.5 – 67.4	80
47.5 – 57.4	70
27.5 – 47.4	50
0 – 27.4	30

6 Academic Integrity

Ensure you read the document Academic Integrity in Coding, available on Blackboard.