University of Newcastle
School of Electrical Engineering and Computing
SENG1110/6110 Programming Assignment 2 – Semester 1, 2019
Due: By electronic submission (Blackboard) by **11:59pm on Fri, 31/05**

ALCOLWORTHS SUPERMARKETS - Part 2

Introduction

The objective of this assignment is to extend the implementation of Assignment 1 using **arrays** and **external files**.
SENG1110 students - This assignment can be completed **in pairs**.
SENG6110 students - This assignment must be completed **individually**.

Before you start

Carefully read the specification below. Make sure you have all the information necessary to start writing the program. If you are uncertain of something, do not make assumptions. Post your questions to the discussion board forum named "assignment 2" and check it regularly. We will add a Q&A document in Blackboard as soon as we have enough questions from you.

Specification

The program will **manage up to 4 depots**. This must be done using an array to hold the Depot objects. Each depot will **hold up to 5 products**. This must also be done using an array.

When run, the program will display a menu of actions to the user, including one that exits the program. Until the user chooses to exit, the menu is displayed again after each action is completed.

The program should have the following functionalities:

1. A user may **add a depot**.
   The user will specify the depot's name.
   There should be an error message if the depot already exists, or if there are already 4 depots.
2. A user may **remove a depot**.
   The user will specify the depot's name.
   There should be an error message if the depot does not exist.
3. A user may **add a product to a depot**.
   Normally, the user will specify the product's name, price, weight, and quantity in inventory, along with the depot's name (price, weight and quantity need to be positive, if not, the program will show a message and ask the input again)
   If, upon being given a name, there is a product in any depot with that name, the program should indicate this. The user will then only be queried for a quantity and depot name. Other details will be taken from the existing product.
      eg: Product <productName> exists, with price $<price> and weight <weight>. Adding additional items.
   There should be an error message if the depot does not exist, or already holds 5 different products.
4. A user may **remove multiple product items at once**.
   The user will specify a product name and a depot name, and a positive quantity.
   There should be an error message if the product does not exist in the depot, or the depot doesn't exist, or the number of products is less than the number to remove.
   Otherwise, the quantity in inventory will be reduced by the specified quantity. After that, if the quantity of the product in inventory is zero, then the product should be removed from the depot. There should be output indicating this.
      eg: <numRemoved> items of Product <productName> removed from depot <depotName>

5. A user may **query for a list of depots**.
   There should be output, describing the depots.
   - Normally, there should be one line per depot.
     - eg: Depot <Depotname> has <number> products
   - If there are no depots, the output should be one line.
     - eg: No depots exist
6. A user may **query for a list of products in a depot**.
   The user will specify the depot's name.
   There should be an error message if the depot does not exist.
   Otherwise, there should be output describing the products.
   - Normally, there should be one line per product.
     - eg: Product <productName> has price $<price>, weight <weight>kg, and quantity <quantity>
   - If there are no products, the output should be one line.
     - eg: No products in depot
7. A user may **query about a product's presence in the depots**.
   The user will specify the product's name.
   There should be an error message if the product doesn't exist.
   Otherwise, there should be one output line for each depot the product is in.
   - eg: Product <productname> is in depot <depotName> with quantity <quantity>
8. A user may **query for the cumulative value of all products in a depot**.
   The user will specify the depot name.
   There should be an error message if the depot doesn't exist.
   Otherwise, there should be output describing the cumulative value.
   - eg: Depot <Depotname> has cumulative product value $<number>

   As an example, let's say there exists a depot D1 containing products P1 and P2. P1 has price $10 and quantity 2. P2 has price $5 and quantity 100. The cumulative value of all products in D1 would be 10*2+5*100 = 520.
9. A user may **export depot and product information to a file**.
   The created file should consist of zero or more lines, each of which should have one of these forms:
   - <depotName> <productName> <price> <weight> <quantity>
     - This indicates that a depot <depotName> exists and holds a product <productName> with the specified details.
     - eg: newcastle-depot apple 1.50 0.1 20
   - <depotName>
     - This indicates that a depot <depotName> exists, but specifies nothing about it's products. It should be output for empty depots.

   An error message should be output if the file is unable to be created, or if a problem occurred during writing.
   An example file (example.txt) is provided on Blackboard.
10. A user may **import depot and product information from a file**.
    The file format should be the same as that used for exporting.
    The imported information is added to the pre-existing depots and products. A "<depotName>" line causes addition of a depot <depotName>, if one does not exist. A "<depotName> <productName> <price> <weight> <quantity>" line normally causes addition of a depot <depotName> if one does not exist, and addition of <qty> items of a product <productName> with price <price> and weight <weight> in that depot. If the product already exists, the <price> and <weight> parts of the line are ignored.
    An error message should be output if the file is unable to be opened, or if a problem occurred during reading. This includes cases such as finding a line with an invalid format, and being unable to add a depot or product due to size limits (in both cases, the program continues to process the rest of the file).

Product names and depot names will be converted to lowercase after input (so a product name "Lava lamp" will be interpreted just like "LAVA LAMP").

Program Requirements

The program should consist of 3 classes:

- **Product** - stores the following details about a group of product items.
  - name - String - the name of the product. Must not contain spaces.
  - price - double - the price of the product. Must be positive.
  - weight - double - the weight of the product, in kilograms. Must be positive.
  - quantity - int - the number of product items. Must be positive.

- **Depot** - stores the following details about a depot.
  - name - String - the name of the depot. Must not contain spaces.
  - products - array of Product with size 5 - the products held by the depot.
- **Interface** - provides the user interface.
  - depots - array of Depot with size 4 - the depots managed by the program.

All data fields of your classes should be **private** (this is imposed so that you apply the principles of **encapsulation**).

Your classes will also need methods to provide the required functionalities. The only class which should have a **main method** is Interface.java, which should create an instance of the class Interface, and call a method run(), which will display the menu to the user. This class will be the only one that takes input from and sends output to the user. It may do so using either TIO or GUI methods (your choice). A template is shown below.

```
public class Interface {
        private void run(){
                //...
        }
        public static void main(String[] args){
                Interface infFace = new Interface();
                intFace.run();
        }
}
```

Marks will be awarded for layout (including visual aspects (variable names, indentation) and structural aspects (variable scope, method usage)), documentation (comments), and the submission's ability to perform as specified. A more detailed marking schema will be made available on Blackboard.

What to submit

You should submit the .java files (Product.java, Depot.java, Interface.java) and an assignment cover sheet, in a compressed .zip file, via the "Assignment 2" link on Blackboard. Do not include .class files in your submission. Add the name of the student(s) on the top of each Java file submitted. If you are completing the assignment as a group (only SENG1110 students), add both names in each Java file AND submit two individual assignment cover sheets OR one group assignment cover sheet.

Extra work for SENG6110 students

For the functionality "query for a list of depots", the output must be alphabetically sorted using the depot names. Similarly, for the functionality "query for a list of products in a depot", the output must be alphabetically sorted using the product names. Extra material about sorting algorithms, including lecture slides and a video, are available in Blackboard (week 9).
SENG1110 students that implement this task will receive extra 15 points (the maximum mark of your assignment 2 is 100)

Late Penalty and adverse circumstances

Note that your mark will be reduced by 10% for each day (or part day) that the assignment is late. This applies equally to week and weekend days. You are entitled to apply for special consideration if adverse circumstances have had an impact on your performance in an assessment item. This includes applying for an extension of time to complete an assessment item. See https://www.newcastle.edu.au/current-students/learning/assessments-and-exams/adverse-circumstances for more details.

Prof Regina Berretta
2019