**University of Newcastle**
**School of Electrical Engineering and Computer Science**
**SENG1110/6110 Programming Assignment 2 – 2016**
**Due:** By electronic submission (Blackboard) by 11:59pm on **Friday May 27**.

## Starberk's coffee replenishment system – part 2

This assignment will use basically the same ideas from the first assignment, but the number of products is not fixed. The user can add products and also delete them. You will also include a feature to load products using an external file.

### Problem specification

Starberk's coffee has asked you to create a system to enable them to manage the replenishment of different products they sell in two different stores (LambtonStore and CallaghanStore). Each product has the following information: name, demand rate, setup cost, unit cost, inventory cost and selling price. Demand rate is a fixed demand per week. The setup cost is a fixed cost that Starberk's coffee needs to pay every time they make an order, regardless of the quantity. The inventory cost is the cost for each product unit in the inventory per week.

The dilemma is if Starberk's Coffee asks for a big quantity order to cover the whole month, for example, they will spend a lot of money in inventory. On the other hand, if they decide to make an order every week they will spend a lot in setup, since every time they ask for an order they need to pay a setup cost.

There is a method called the EOQ (Economic Order Quantity) which calculates the optimal order quantity, if you know the following information:

Demand rate
Inventory cost/unit/week
Setup cost

The EOQ method calculates the optimal order quantity that would minimize total variable costs, by using the formula below

$$Q = \sqrt{\frac{2sd}{h}}$$

Where s = setup cost
d = demand/week
h= inventory cost/unit/week

An example:
Suppose that you know the demand rate for coffee is 45/week, h = 0.60 (inventory cost per unit per week) and s = 132 (setup cost). With these values, you can calculate the value of Q which will be 140.71. Suppose that you can just order integer values, so rounding 140.71 we will have Q=141. Now, you can find the replenishment strategy for the next n weeks. Let's suppose 6 weeks. In the first week the order will be 141 units, but the demand is only 45 units, then it will have 96 units in the inventory. In the second week the demand is 45 units and we have 96 units in the inventory from week 1. So, the second week will finish with 51 units. Then week 3 will finish with 6 units in the inventory. By week 4, there is not enough units in the inventory, so we need to make a new order, and so on. So, the replenishment strategy for next 6 weeks will be

| Week | Quantity Order | Demand | Inventory |
|---|---|---|---|
| 1 | 141 | 45 | 96 |
| 2 | 0 | 45 | 51 |
| 3 | 0 | 45 | 6 |
| 4 | 141 | 45 | 102 |
| 5 | 0 | 45 | 57 |
| 6 | 0 | 45 | 12 |

Notice that the inventory finishes with 12 units. This is not necessary; it is better the inventory finishes with 0 units. So, we can do some arrangements. Just change the last order from 141 to (141-12). So the best replenishment strategy for next 6 weeks will be

| Week | Quantity Order | Demand | Inventory |
|---|---|---|---|
| 1 | 141 | 45 | 96 |
| 2 | 0 | 45 | 51 |
| 3 | 0 | 45 | 6 |
| 4 | 129 | 45 | 90 |
| 5 | 0 | 45 | 45 |
| 6 | 0 | 45 | 0 |

Now, let's calculate the total cost and the total profit. Suppose the unit cost is 1 and the selling price is 3. The total cost will be

*setup* *unit cost*

Purchase = (132+132) + (141+129)*1 = 264+270 = 534
Inventory = (96+51+6+90+45)*0.6 = 172.80
Total cost = 534+172.8 = 706.80

The profit will be

Profit = 45*6*3 – 706.8 = 810 – 706.8 = 103.20

For this assignment, you will write a program that will calculate and print the replenishment strategy for a product for **n** weeks and find the total profit. In the end, the program will ask the user if he/she would like to restart for another product.

**Programming Requirements**

When the program starts, a 2 tier menu system will be available, where the user will have the following menu options:

1. **Choose Store**
   **{callaghan, lambtom}**
   1. **Add/Edit product**
   2. **Delete product**
   3. **Display product**
   4. **Display all products**
   5. **Exit Store**
2. **Display stores**
3. **Open**
4. **Save**
5. **Exit**

The description of each option is below.

- **1. Choose Store**. The user chooses a store (Lambton or Callaghan). The input name must be a string from the set {callaghan, lambton}. The user can use lower case or upper case, but the program will always convert to lower case. If the user inputs a string that does not belong to the set, then the program will show an error message and ask the input again. Next, the user will have the options below.
  - **1. Add/Edit product:** input data for one product. The program will ask the name of the product. The user can use lower case or upper case, but the program will always convert to lower case. It will also be possible to have space within the name of the product (such as “Tic Tacs”). If the user inputs a name that already exists, then the program will show a message and the user will have 2 options: (1) input another name or (2) change the data of this product.
  - **2. Delete product:** If the store does not have any products, the program will display the message “no products” and go back the second level menu options Add/Edit product, Delete product, Display product, Display all products. If the store has products, the list of product names will be displayed and the user will be asked to input the product name. Then, the program will delete the product and will display the message “the product was deleted”. If the name is not found, the program will display the message “the product does not exist” and go back to the second level menu options.
  - **3. Display product:** If the store does not have any products, the program will display the message “no products” and go back the second level menu options Add/Edit product, Delete product, Display product, Display all products. If the store has products, the list of product names will be displayed and the user will be asked to enter the product name. If the name is not found, the program will display the message “the product does not exist” and go back to the second level menu options. If the name is found, the program will display all information about the product (name, demand, setup cost, unit cost, inventory cost and selling price), then, the program will ask if the user would like to see the replenishment strategy. If yes, the program will ask the number of weeks and it will display replenishment strategy, total cost and profit. If the name is not found, the program will display the message “the product does not exist”.

    ***Note****: for some inputs, the EOQ method will not give a feasible plan. For example, if setup cost is 5, demand rate is 100 and inventory cost is 1, then the Q = 31.6, which does not cover even the demand for the first week. If the value of Q is not a feasible value (if it is less than the demand rate), then the program will give an error message to the user saying “It is not possible to have a replacement strategy with the inputs given, please edit the product details” and go back to the second level menu options.*
  - **4. Display all products:** If the store does not have any products, the program will display the message “no products” and go back the second level menu options Add/Edit product, Delete product, Display product, Display all products. If the store has products, the program will display the following information (name, demand, setup cost, unit cost, inventory cost and selling price) of each product. The details should be displayed neatly in columns, with a header row displaying the content for each column.

- **5. Exit Store**. The program will go back to the options choose store, display store, open, save, exit.

***Note:*** *Data validation is required for any input provided by a user. All product names must be within 3 and 10 characters in length. All number inputs must be positive. If the user inputs an invalid product name or negative input, the program will show a message and ask the input again*.

- **2. Display Stores:** The program will display information about the stores as in the example below:

```
Store: Lambton
        Number of products: 0
Store: Callaghan
        Number of products: 2
        Product 1: name
        Product 2: name
```

***Note:*** *if products exist, only the product name needs to be displayed.*

- **3. Open**: This option will load products to a store via batch uploading/file reading. The file can contain information for each store. For this option, the user needs to input the file name. If the file does not exist the program will display the message "the file does not exist" and will go back to the main menu, choose store, display store, open, save, exit. If the file exists, the program will read the information of the products in this file and add them to the list of exisitng products (if any) to the required store. You will find an example input file named "Input.dat" in Blackboard. You can assume that any files used in the program will match the format of this provided file. You can also assume that any product details in the file will contain valid data, so data validation will not be required for this option.
- **4. Save:** The user will input a file name (name only) and the program will save the details of the store to a file with a .dat extension. The program will save any information which is current for the store, in the same format of "Input.dat". If the user enters "Input" as their file name, then the details in the system will overwrite those in the file.
- **5. Exit**. The program will end.

To write this program, there must be three classes: Product, Store, StarberksInterface

**The Product Class** (the file needs to be **Product.java**)

It will hold the required instance data for a product and it will have suitable methods to access and modify the data for a product. It will also have a **toString** method. The methods will be public and the instance variables will be private. Implement two constructors; one default constructor and one paramatized constructor.

**The Store Class** (the file needs to be **Store.java**)

It will hold the required instance data for a store and it will have suitable methods to access and modify the data of a store. It must have a one-dimensional array of Product as an instance variable. It must use the Product class's methods for accessing and modifying the data in a product object. The product array will be instantiated with 3 positions. You need to use a method resizeArray to add a position for a new product when necessary. ***This class should not use libraries*** for any array functionality. The class will also have a **toString** method. The methods will be public and the instance variables will be private.

**The StarberksInterface Class** (the file needs to be **StarberksInterface.java**)

Creates the interface (TerminalIO or GUI, you choose).
This class will have two Store objects as instance variables: LambtonStore and CallaghanStore
This class will receive all input from the user, display all output, check for invalid inputs and display all error messages. **No other class should have GUI or TIO operations**. That is, the class **StarberksInterface** will be the only one that will receive inputs and show outputs.

All the data components of all your classes need to be **private** (this means that you are applying the principles of ***encapsulation***). Additionally, your classes need to have methods that provide the functionality outlined in the problem description. The only class which should have a **main method** is **StarberksInterface.java**, which should create an instance of the class and call the run() method which will have code to provide the user with a menu system to allow them to perform any of the tasks outlined in the problem description. The template for this is below.

```
public class StarberksInterface
{
	private void run()
	{
		//This method should control the flow of the program
	}
	public static void main(String[] args)
	{
		StarberksInterface intFace = new StarberksInterface();
		intFace.run();
	}
}
```

**What to submit.**

You should submit the Java files (Product.java, Store.java, StarberksInterface.java), the documentation and the assignment cover sheet electronically using the Blackboard. Your code also must be very well commented and organized.

For submission, please zip the folder you wish to submit, named c<studentNumber>_SENG1110_Assignment2
Example:

cXXXXXXX_SENG1110_Assignment2.zip

Files which will need to be included with the submission are:

Product.java
Store.java
StarberksInterface.java
Assessment Coversheet

**Extra Work for SENG6110 students**

When the user chooses the option “display all products”, the program will display the information sorted by name or demand (the user will choose). You need to implement a sort algorithm.
**SENG1110 students that implement the SENG6110 task will receive extra marks**.

**In the Blackboard you will find a new forum in the discussion board: “Assignment2”. Any question about the assignment 2 you can post there. Check this forum regularly.**

Dr. Regina Berretta
April-2016