University of Newcastle
School of Electrical Engineering and Computer Science
SENG1110/6110 Programming Assignment 1 – Semester 1, 2016
Due: By electronic submission (Blackboard) by 11:59pm on **Friday April 22**

Starberk's coffee replenishment system

Introduction

The objective of this assignment is the implementation of an object oriented program using Java.
Note: Assignment 2 will be an extension of assignment 1.

Problem Description

Starberk's coffee has asked you to create a system to enable them manage the replenishment of different products they sell in one specific store. 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 and the unit cost is the price per unit. The inventory cost is the cost for each product unit in the inventory per week.

The dilemma is if Starberk's Coffee places a large order to cover the whole month, they will spend a lot of money in inventory. If they decide to make an order every week they will spend a lot in setup, since every time they ask an order they need to pay a setup cost.

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

Demand rate
Inventory cost/unit
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 that 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.

Program Interface

When the program starts the user will have the following menu options:

1. input data for one product
2. show data from one product
3. show the replenishment strategy for a product
4. exit program

If the user chooses **(1) input data for one product**, then the program will ask for the name of the product. The input name must be a string with size between 3 and 10 characters (if not, the program will show an error message and ask for the input again). The user can use lower case or upper case or a mixture of both, but the program will always convert to lower case. While the program is running, if the user inputs a name that was already provided, then the program will show a message and the user will have the option to change the name or change the data in this product. For the other inputs (demand rate, setup cost, unit cost, inventory cost and selling price), if the user inputs a negative number, the program will show a message and ask for the input again. When the input finishes, the program will return to the menu (above). If a user tries to enter a product, and three products already exist, a message will be displayed informing the user that there is no more room in the product database. The user will then have the option to remove a product or return to the main menu. If a product is removed, the user will be able to enter details to add a new product.

If the user chooses **(2) show data from one product**, and there is no product data, the program will display a message and return to the main menu. If product data is available, then the program will ask the user for the name of the product. The user can use lower case or upper case or a mixture of both when providing the product name. If the user inputs a product name that does not have data or does not exist, then program will show an error message and ask the user for input again. When the user has provided an existing product name, the product data will be printed (name, demand rate, setup cost, unit cost, inventory cost and selling price). Next, the program will return to the menu (above).

If the user chooses the **(3) show the replenishment strategy for a product**, if there is no product data, the program will display a message and return to the main menu. If product data is available, then the program will ask the user for the name of the product. The user can use lower case or upper case or a mixture of both when providing the product name. If the user inputs a product name that does not have data or does not exist, then program will show an error message and ask the user for input again. Next, the program should ask the number of weeks. Finally, the product replenishment strategy will be printed. Next, the program will return to the menu (above).

If the user chooses the **(4) exit program**, if there is no product data, the program will exit. If product data is available, the program will output the best replacement strategy, which will be the product with the highest profit. You do not need to print the replenishment strategy, just the name of the product and its profit.

Note that 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 even cover the demand for the first week. If the value of Q is not a feasible value (if it is less the demand rate), then the program will give an error message to the user saying that is not possible to have a replacement strategy with the inputs given and ask the user inputs that data again.

There must be three classes: Product, Store and StarberksInterface:

The Product Class
The file name needs to be **Product.java**
This class will hold the required instance data for a product and it will have suitable methods to access and modify the data for a product.
This class should have two constructors.

The Store Class (the file name needs to be **Store.java**)
It must have three Product objects as instance variables: product1, product2 and product3.
It must use the Product class's methods for accessing and modifying the data in a product object.

The StarberksInterface Class (the file name needs to be **StarberksInterface.java**)
Creates the interface (you can use TerminalIO or GUI, your choice).
It must have one Store object as instance variable.
This class will act as the interface for the program, meaning it will receive all input from the user, display all output, and check for invalid inputs & display all error messages.

No other class should have GUI or TIO operations

Other considerations
All 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 StarberksInterface and call the run() method. The run() method will have the code to provide the user with a menu to allow them to perform any of the tasks outlined in the problem description. The template for how to implement this feature is provided 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();
	}
}
```

Remember: The class **StarberksInterface** will be the only one that will receive inputs and show outputs.

Your program should display all information such that is easy to read. Your code also must be very well commented. You should not use arrays. Your solution should be your own. A marking schema is available in Blackboard.

What to submit.
You should submit the Java files (Product.java, Store.java, StarberksInterface.java, assessment cover sheet) electronically using Blackboard. For submission, please zip the folder you wish to submit, and name it c<studnetNumber>_SENG1110_Assignment1. For example cXXXXXXX_SENG1110_Assignment1.zip.

Extra Work for SENG6110 students
You need to describe the classes using UML class diagrams.

In Blackboard you will find a new forum in the discussion board: "Assignment1". If you have any questions about the assignment 1 you can post there. Check this forum regularly.

Dr. Regina Berretta
Apr-2016