Thank you!

Grade / ???

SYSC2004C
Object Oriented Software Development

Milestone 1

Date Due: 10-02-2021

Time Due: Before 20:00 EDT

Student: Gabriel Benni K. Evensen [101119814]

Student: Ngo Huu Gia Bao [101163137]

- The Product class.
- The Inventory class.
- The StoreManager class

---

1. **Questions:-**

   (a) What is a constructor? When does it get used in Java? **[3 marks]**

   A *constructor* is a method which has the same name as its respective class. Constructors are used for initializing an object in said class. A constructor is called when an object of that respective class is created.

   (b) Did you need to specify a constructor for the StoreManager class? Why or why not? **[3 marks]**

   We chose not to specify a constructor for the StoreManager class. This is because, although it might be helpful if we decided to give the store a variable name, it is not necessary, and it may just end up complicating the code, or worse yet; being "spaghetti code". It could also be useful if we decided to declare an inventory object inside of the constructor, although this has not been explicitly asked, so in sticking with the guidelines, we have not done so.

   (c) Explain what a default constructor is.

   A *default constructor* is the constructor that either have or not have the parameter, and which initializes the uninitiated instance variables to their default values (`null`). For example:

   ```
   public Product() {this(null);}
   ```

   (d) What are object references and where are they used in this milestone?

   Object references are the objects that are created on the heap of their respective classes, and point to that class. An object reference may access all methods and attributes that are in their class. For example:

   ```
   Inventory inventory1 = new Inventory;
   ```

   Here, inventory1 is the object reference of the Inventory class which allows the users to access to the data or behaviours that were encapsulated.

   (e) Summarize the most important differences between an ArrayList, LinkedList, and Array. Which one did you use in this milestone and why? If you used something else, you must explain why as part of your Change Log.

   ArrayList:- An ArrayList may only store the object, not the primitive types, and the size will grow and shrink dynamically. Adding new items to an ArrayList helps to increase efficiency when there is no need to update the capacity or modify the index. We can print out the entire ArrayList and access any indices easily. ArrayLists are efficient when adding or removing elements at the end.

   LinkedList:- A LinkedList is efficient for fast removal but slow for iteration. LinkedLists are efficient when looping through indeces, as opposed to accessing random indeces. Adding or removing items from the beginning, middle, or end, is easy with LinkedLists.

   Array:- An Array has a fixed Size (i.e., it is static), set upon declaration, or initialization; differentiating

it from an ArrayList. We may access any index from the Array, as per our choosing. The Array cannot be printed out.

For our project we chose to use a HashMap. HashMaps allows for the storing different value types, while an ArrayList cannot do the same.

(f) What is encapsulation and how is it relevant to this milestone?

Encapsulation is the act of abridging the data and behaviour into their respective object.

Encapsulation makes the interface look simpler by hiding the variables from the users. In this milestone we make an interface where users can see the product information. Encapsulation is used here primarily because we do not want the users to modify the product information such as, name, price, and id.

2. **Change Log:-**

(a) StoreManager Log:-

For the **StoreManager** we chose to make *checkStock void* since having to manually type `System.out.println(...)` each time you wish to check the stock could get tiring, very quickly. Additionally, the wording of the guideline makes it seem as this is a method the user would call when they wish to check how much of an item is left in the stock, thus this is the easier implementation for them.

We chose to make the *processTransaction* method have a parameter consisting of a 2D array. This 2D array is then converted into a hashmap (of the same length as the array) using one of the iterator methods learned in Lab 3, and simple array operations. The variable *total* keeps track of the cumulative running total price (in an unspecified dollar currency) of the items so far. We use a simply `if` statement to check if there is adequate wares for the transaction to take place, **iff** there is, then the *total* will be incremented.

(b) Inventory Log:-

We chose to declare and initialize products in the *Inventory* constructor for now. Although, we are hoping to eventually be able to add products from the terminal, and have them dynamically added to the inventory. This implementation plan would ideally not change our StoreManager class much.

For *gettingQuantity* we decided to return an int since we are returning the number of items within the inventory. Using the knowledge gained from Lab 3, we have implemented an iterator, that looks for a specific key, within the keyset. If a match is found, we get the quantity, and return it. If there is not a match found, we simple `return -1`.

(c) HashMap:-

We decided to choose HashMap because HashMap can store different type of values which can be easily to access and get. In addition, we do not need to looping through the entire HashMap like ArrayList to find a specific value since there is the support syntax **.containsKey()**. Finally, we can add new items to the HashMap effective and quick with the syntax **.put(key,value)**