SYSC2004 – Project Report
Milestone 1

Sebastien Marleau
SN: 101155551

Questions

1. **What is a constructor? When does it get used in Java?**

A constructor is a class method of a Java class responsible for initializing an object of that class. It is used to initialize any object in Java. If a constructor is not provided by the coder for a class, a default constructor will be used to initialize it.

2. **Did you need to specify a constructor for the StoreManager class? Why or why not?**

The StoreManager class cannot use the default constructor due to the need to initialize its Inventory instance variable. If a custom constructor was not provided, the Inventory instance variable would stay null, and any instance methods trying to use this variable would cause an exception to be thrown.

3. **Explain what a default constructor is.**

Custom classes always have a superclass. This is by default the Object class, but can be specified to be any other class. When no constructor is provided by the coder of a custom class, Java adds a no-argument constructor to this custom class who's only purpose is to call the no-argument constructor from the direct parent in the class hierarchy (the superclass).
This means that no variables declared in the custom class will be initialized by this default constructor, but those of the superclass will.

4. **What are object references and where are they used in this milestone?**

Object references contain the address in memory where all the variables and methods of an object reside. It makes it possible to call methods on that object from anywhere in the code, as long as you have the reference.
Object references are used every time objects are created and stored in variable, passed as method parameters, or called a method upon. As in Java almost everything is an Object apart for primitive data types like ints and doubles, we use many object references in Milestone 1. Here is a list:

- **Product**
  - The name and ID variables both have a reference to a String object
  - Used to keep track of these fields in a human-readable format, and transmit this information easily to any place in the code that needs it
- **Inventory**
  - Has 2 HashMap object references as instance variables
  - Uses String object references as parameters to change code behavior based on the content of the strings
  - Uses the HashMap references to modify the content of these HashMaps to enable a dynamic inventory system where adding or removing different Products and their stock quantities is possible
- **StoreManager**
  - Has an Inventory object reference as an instance variable
  - Uses the inventory object reference to get information and modify this Inventory instance.

5. **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.**

Arrays vs ArrayList:
ArrayList itself uses Arrays to store data. It is basically an array but at a higher level, where you don't have to keep track of the size and capacity of the array, and having to make the Array bigger once it gets full. This makes it very useful for most cases where you have a changing quantity of something. It is slightly less efficient than an Array though, since it sometimes has to allocate a bigger Array and copy everything to it.

LinkedList vs ArrayList:
A LinkedList offers the same functionality as an ArrayList, but does not use Arrays under the hood. It uses nodes, where the first item in the list points to the next, and so on. This makes it very efficient at inserting or removing elements anywhere in the list, whereas ArrayList or Arrays would do this slowly due to having to shift every item in the list. LinkedList is very inefficient at finding the i^{th} element in the list though, having to iterate through i elements before getting to it. It also requires more memory due to having to store pointers to the next elements for each element in the list.

Implementation in milestone:
I used a HashMap for the Inventory class in this milestone. The key to value relationship represented what was needed of this class. It was described that information of the product or their stock quantity had to be available from a product ID. This was easily implemented using this product ID as the key and the value being the information needed, requiring 2 HashMaps for full functionality.
It would have been possible to implement this similarly using an ArrayList, but the search functionality would have been less efficient due to HashMaps using Hash tables and Arrays having to search sequentially. Furthermore, the organization may have been more confusing. 2 Arrays would have had to be used, where one stores the products and the other the stock quantity. It would be easy to make an error with the relationship between these two array.

6. **What is encapsulation and how is it relevant to this milestone?**

It is the idea of hiding implementation details and limiting access to attributes from the user. This can help reduce bugs, simplify code, and produce modular code. A relevant example in this milestone is the Inventory class. The user does not need to know the implementation uses HashMaps to keep track of the products and stock. This allows the Inventory class to maybe change to and ArrayList implementation in the future, without having to change any other code. By keeping the public interface to what an inventory should be able to do, it also allows this inventory class to be used in another project where an inventory is necessary.

Changelog:

Product:

- ➔ Created a basic class with 3 fields that only have getters, no setters.
- ➔ The fields are final due to there being no setters, they will never be changed after inintialization.
- ➔ They are private and not protected due to the child classes probably not needing to access (or at least modify) these object variables anyways.

Inventory:

- ➔ Uses two HashMaps
  - One with key: ID and value: product stock
  - The other with key: ID and value: Product object
  - Using two maps is undesirable, but the specification said we were not allowed to add an extra variable to the Product object, which is where I would have put the stock
  - The only other option would have been to create an extra class which only has a Product instance and a stock quantity
    - These would have had to be initialized every time a new Product is added, which would have made the implementation a bit more complex
  - HashMaps provided cleaner code and better efficiency in their search function compared to ArrayLists
  - They also represent the purpose of the inventory better than the ArrayLists would have
- ➔ Stock quantity managing:
  - I created only a single function to add and remove stock of a product. This is due to their being a lot of repeated code if this was separated into two functions.
  - The function throws an exception when the user attempts to remove more stock quantity than there is available. This better represents the failure compared to a returned boolean or negative integer, in my opinion. It is also something that could and should have been checked in advance.
- ➔ Product information acquiring
  - The specification was not very clear on what sort of information the Inventory was supposed to return, so I decided to just make it return the Product object. That way the user can get the information they need without complicating the Inventory class.
- ➔ Get stock quantity
  - I decided to have the function return 0 on a request where the Product is not registered in the library. I could have thrown an exception, but it makes more sense that if the product is not in the inventory, there would be none of it (hence a return of 0).
- ➔ Getting a product from ID
  - This function return null when the product ID is not in the inventory. Could have thrown an exception, but I felt like this was the kind of error the user would anticipate and code towards. Having a null return value makes that easier than having a try catch.

StoreManager:

- ➔ performTransaction:
  - Had to create an extra class that wraps product IDs with quantity to be able to pass this information as an array to the method. This is probably not how the implementation would work in future milestones
  - The method checks beforehand to make sure the inventory has the entire quantity of products available before starting to remove them from the inventory. This is to make sure on a failure the transaction does not remain half-complete
  - It return -1.0 instead of an error on failure due to the fact that the error should be anticipated