**Milestone 1 Report**

Gilles Myny (101145477)

Tawseef Patel (101145333)

**Questions:**

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

Constructors in Java are used to initialize objects, they can be empty or carry information to set initial values. Constructors are automatically generated if the user does not manually define one themselves. It does not have a return type but can include parameters.

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

Yes, a constructor was created for the *StoreManager* class, it does not contain parameters or attributes. The manual creation of this constructor was not needed but would have been created automatically. It is used when creating a *StoreManager* in a test class to ensure the code runs without errors.

1. Explain what a default constructor is.

A default constructor is provided by the system when the programmer does not specify one themselves. It will call the constructor of the superclass by default. It is good practice to define a constructor manually even with no parameters or attributes are inserted.

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

Object references are a link to an object. Variables cannot contain objects but only references to them. Objects are constructed and allotted in memory separately from the declarations of variables. Objects denoted by a literal are allocated in memory at compilation time and all others must be constructed and allocated through explicit statements. Essentially, an object reference is when you create an object and initialize it, and then create another object and assign it to the first object, you can update the first object and have the second object be updated. An object reference that is used in this milestone is clear. We use it in the Product Class through the use of this keyword. The *this* keyword is a reference to the current object and can refer to any member of the current object from within an instance method or a constructor.

1. 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*’s are the most flexible and easy to use list type when it comes to constant manipulation of data within the ArrayList. By automatically resizing itself it eliminates the need to check for OutOfBound errors as seen when dealing with regular *Array*’s. In a *LinkedList* each element is a separate object with its own address making it easy to move between indexes using pointers, therefore increasing the speed of data access. Both *ArrayList*’s and *LinkedList*’s are resizable, unlike the *Array*. An *Array* is a fixed sized data structure which is defined when initializing the array. The important difference between *ArrayList*’s and *Array*’s is that arrays allow objects of a class as well as primitive data types unlike *ArrayList*’s which only allow the former. Two *ArrayList*’s were used in this milestone to store products and stock information. The reason they were used is to allow for the most flexibility to alter and manipulate these two sets of data with ease. Not having to worry about pointers from a LinkedList or sizing issues from a regular Array is crucial to the efficiency of the code for the project at hand.

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

Encapsulation means to encapsulate the data and behavior into a single object. Single object contains both its data and behaviors and can hide what it wants from other objects. It is the mechanism of wrapping data and the associated methods. It can also be known as data hiding because you declare the variables of a class as private and provide public setters and getter statements. Getter and setter methods are the methods which can be used as the access points of the instance variables. This is relevant to this milestone because the StoreManager needs to be able to access the private instance variables in the Product and Inventory classes, but this can only be done if the getter and setter statements are made public. Thus, we used getter and setter statements to add inventory into an ArrayList to better manage and access the relevant data.

**ChangeLog:**

1. (2021-01-04)

Release Highlights

* Product.java class
* Inventory.java class
* StoreManager.java class
* UML diagrams for all 3 classes

Features

* Product.java: allows StoreManager to create and set a name, ID, and price for the product. Includes getters and setters for required public information. Creates a print override for proper printing of each product to console. Checks for duplicate products.
* Inventory.java: allows StoreManager to create a new inventory. Contains two ArrayList’s for products and stock information. Setters allow StoreManager to create (if the product does not exist) or add and remove product stock given the product name or the product ID. Getters allow StoreManager to get product info with the product ID, returns the price of a quantity of items given the product ID, or get stock amount with the product ID.
* StoreManager.java: creates new Inventory and a variety of products given their name, ID, and price. Tests adding the created products to the inventory with stock amount; restocking products given their ID’s with amount to restock; removing stock of products given their ID’s with amount to remove. Returns a Boolean value if all tests are successful.

**Report Written By: Gilles Myny (101145477)**