SYSC2004 – Project Report
Milestone 5
Sebastien Marleau
SN: 101155551

Change log:

- ➔ Renamed ProductQuantityCollection to ProductStockContainer and renamed methods to fit the milestone requirements
  - Did not follow the input type requirements on the methods, as I wanted to keep the productID abstraction between the StoreManager and the containers
  - Did not make ProductStockContainer abstract or an interface, but rather a superclass of Inventory and ShoppingCart. I feel like this model better represents the relationship
    - It makes sense for a ProductStockContainer to be instantiable, and so not be abstract. Such an instance would do just as the name suggests: act as a container for products with their related stock quantities.
    - ShoppingCart and Inventory are both types of ProductStockContainers with only very minor differences. Hence they override a few methods, adding behavior on top of the superclass' methods.

Questions

1.What is the difference between an interface and abstract class?

An abstract class already implements some methods and instance variables. It then marks some methods as "abstract" which must be overridden by sub-classes. Abstract classes act as a superclass, and must be inherited.

An interface could be described as an abstract class where no instance variables are defined, and all methods are marked as "abstract". Interfaces though do not act as a superclass, but rather as a contract. Classes can "implement" multiple interfaces, and this just means they implement all methods set by the interfaces.

Both interfaces and abstract classes can be used as sub-types.

2.Why we should "code against"/"program to"an interface(in relation to OOP)?

Interfaces are used for common behavior across classes that have very distinct hierarchy trees. For example, a car and a plane can both move, and it may be useful to group such "movable" objects together. Without interfaces, you would have to make cars and planes extend a "movable" subclass. This may work, but what if planes want to share flying with birds. This would make a very complex and nonsensical inheritance tree.

Having interfaces allows us to group cars and planes under a "movable" interface, and planes and birds under a "flying" interface without them having to share a single line of code or a common ancestor in the inheritance tree.