MILESTONE 5

Overview

You are going to be putting the *final* finishing touches on your project in this milestone.

*Keep in mind that most of the tasks you will be required to do are relatively **open-ended**. You must justify any decisions you made that were not obvious in your Change Log. Refer to the examples in Milestone 1.*

Final Touches

The store is now complete. However, there are some things you can likely do to make your code better. Someone may want to add some features down the line, and we want to make it as easy as possible for them to do so!

To do this, we should make the specifications for working within our store clear. One way to do this is to enforce certain procedures, while providing a blueprint for following these procedures. This is one of the major purposes of interfaces and abstract classes! Then, when a new programmer comes in and wants to add some new functions, they can use the pre-existing interfaces you set up and implement the minimum required behaviour for their new features to work.

ProductStockContainer

This is a new interface/abstract class will define some basic behaviour that needs to be part of any class that wants to manage a Product-Stock collection. Since any class that wants to manage Products and Stocks must have methods to remove, add, get, etc., it makes sense to define a set of rules for doing this within your source code.

Requirements:

1. You must make a ProductStockContainer abstract class/interface.
    a. Both Inventory and ShoppingCart **must** extend/implement ProductStockContainer.
2. ProductStockContainer **must** have the following method *declarations (declaration does not imply behaviour)*: The _ denotes that if you make an abstract class instead of an interface, you *may* have an `abstract` declaration there.
    a. `public _ int getProductQuantity(Product)`
    b. `public _ void addProductQuantity(Product, int)`
    c. `public _ boolean removeProductQuantity(Product, int)`
        i. **or** `public _ void removeProductQuantity(Product, int)`
        ii. **or** `public _ int removeProductQuantity(Product, int)`
    d. `public _ int getNumOfProducts()`
    e. You can define more but must justify any additions in your Change Log.
3. **You must justify the choice of either interface or abstract class in your Change Log**. Keep the object-oriented principles in mind. Go for code reuse, but do not forego abstraction, encapsulation, the goal of programming to an interface, etc.

4. The unique attributes of ShoppingCart and Inventory must remain intact, e.g., ShoppingCart should still store its cartID.
5. Feel free to use overrides, overloads, *where justifiable*.
6. You must update your UML accordingly.
7. Your store must work exactly the same as before. If it was not working, it must start to work now!

Questions

1. What is the difference between an interface and abstract class?
2. Why we should "code against"/"program to" an interface (in relation to OOP)?

Milestone 5 Deliverables

1. The following classes completed according to Milestone 5 specifications: *StoreView.java*, *ShoppingCart.java*, *Inventory.java*, *StoreManager.java*, *ProductStockContainer.java*, and *Product.java.* Include any resources/extra classes required by your application. All the project files should be zipped into an archive.
2. **Do not** include your JUnit tests.
3. Everything applicable to Milestone 5 from General Submission Requirements. Do not forget the updated UML diagram, to document all your classes, and the report with the Change Log and the answers to the questions.