Isabella Nguyen – 1009051482

CSC207 – Week 7 Design Question

Chosen Design: Design 2

How does this design adhere to SOLID?

S – Single Responsibility Principle

The system only changes when someone wants to make a BubbleTea. If someone wants to customize their own BubbleTea then it will not affect if they want to order a StandardDrink as well.

O – Open Closed Principle

The Item class displays this principle well as the Item interface as the system can easily add new toppings or ingredients without having to rewrite the Item class. Hence, the item is open to extension and closed to modification as the principle states.

L – Liskov's Substitution Principle

Since both Topping and Ingredient implement the Item interface, they should both be able to use all the methods in the Item interface according to the principle. Both classes can use all the methods, therefore they adhere to the LSP.

I – Interface Segregation Principle

The class only implements one interface and none of the client code is implementing any methods they do not require. Therefore, the system adheres to the ISP.

D – Dependency Inversion Principle

Since the Builder class utilizes the item interface rather than the concrete Ingredient and Topping classes, there is no dependency between the Builder class and the two concrete classes. This allows for changes to be made to the Ingredient and Topping classes without having to change more than the classes themselves.

The class satisfies the specification well as the main goal of the system to allow the user to choose from a preset menu using the StandardDrinks Class or customize their own which is done with the Builder class. This class implements the preset drinks menu better as the user can just pick from the standard drinks, however for Design 1, the user would still need to build the standard drinks. Since the StandardDrinks are in a separate class, the code can easily be modified to add more drinks without affecting the rest of the code. In addition, the user can easily add toppings and ingredients without it affecting how the overall drink is made and Item interface helps keeps all the customizations organized rather than having to separate them by Toppings or Ingredients.

Isabella Nguyen – 1009051482

To improve the SOLID principles, I would make the TeaFlavor implement the Item interface as it utilizes the same methods getName() and getCost(). This adheres to the LSP since the tea flavour would not be implementing any unnecessary methods and the rest of the program would not be impacted. This would better adhere to the DIP as the builder does not need to depend on the TeaFlavour. This decouples the system and now more changes can be made to TeaFlavour without affecting how the rest of the program works. With the current system, if TeaFlavour ever needed to be replaces then the logic in Builder would need to be replaced.

An improvement to the specification would be to implement a method that would allow users to adjust their sweetness and ice levels. Most bubble tea places allow you to customize how sweet you would like to make your drink (0%, 30%, 50%, 70%, 100%) and how much ice you want in your drink (No Ice, Less Ice, Regular Ice). This typically comes at no additional cost to the user, so the BubbleTea class would need more attributes such as IceLevel and SweetnessLevel. Then the Builder class can implement another method that would allow the user to pick their ice and sweetness adjustments. This would make the system more curated to the problem domain.