University of Dhaka

Department of Computer Science and Engineering

CSE-3216 – Software Design Patterns Lab

Lab 4

Submitted by:

Deeparghya Dutta Barua, Roll - 28
Jahir Sadik Monon, Roll - 32

Submitted to:

Dr. Md. Samiullah

Assistant Professor

Department of Computer Science and Engineering

University Of Dhaka

Date of Submission: 9th September, 2022

Assumptions:

During our implementation of a solution for the given problem, we had to make some assumptions. Here is a list of the assumptions we had to make –

1. The user is to input money in cents. That is, a single dollar bill will be entered as "100" (without the quotes and so forth).
2. The user is to use the specified delimiter for batch money entry. Using any other character might cause unintended effects.
3. Unstocking implementation of a drink (or any product) is decoupled from the base type of the product. This allows defining different protocols for different quantitative contexts.
4. For now, only drinks have been implemented but the current structure allows adding products that belong to a different paradigm.
5. The vending machine will have sufficient coins of any allowed unit to return the change.
6. Transactions are atomic, and there will be no intermediate state involving discrepancies between the product stock and the balance.
7. The menu of the product listing will strictly follow a "OptionNum. ProductName" format. Any other choice will require changes in the regex.
8. All inserted money is to be ejected when the product is not in stock.

Design Patterns:

Here is a description of places where we have used the given design patterns –

Singleton Pattern: As we had assumed that the TransactionHandler and ProductDispatcher classes should only have one instance throughout the execution of the program, we made those classes Singleton classes and have used the getInstance() method to return the initialized instances.

Abstract Factory: We have implemented the abstract factory pattern by making a common interface IProductFactory, which defines the getProduct() method that creates a product. Specific factory classes, DrinkFactory in this case, implements this abstract factory interface and returns IDrinks. Using this abstract factory interface, we can add other types of styles that have different requirements and constraints to our system without disrupting the previous code.