Technical documentation

General Overview

The system was designed using C++ through a Visual Studio solution; SupermarketCheckout.sln. The solution contains 3 different projects:

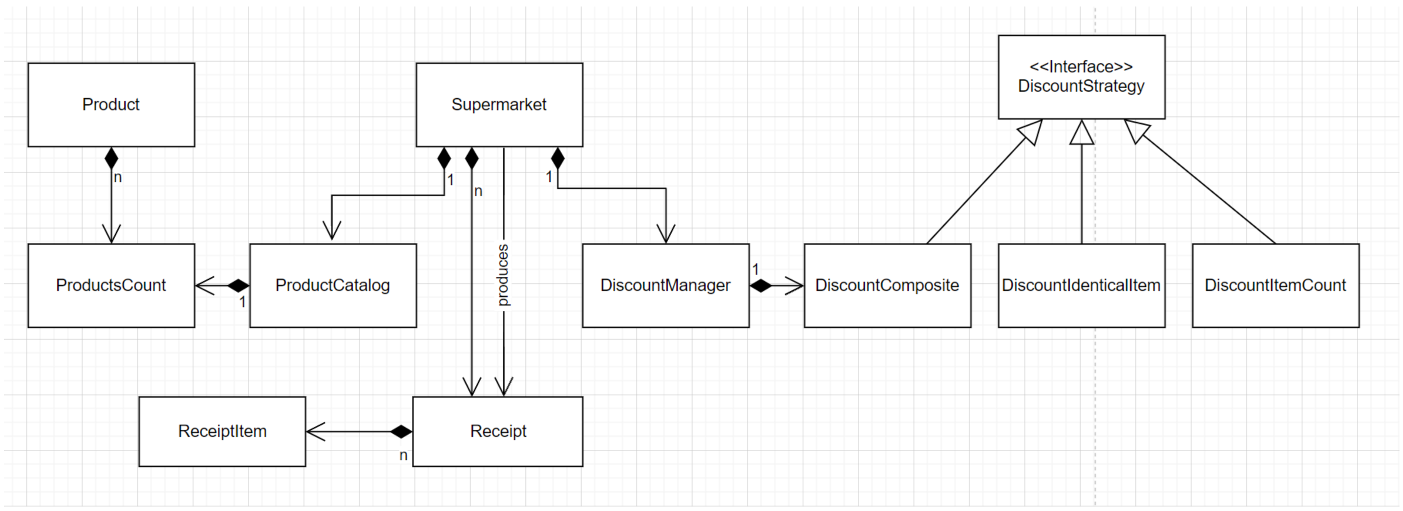
- SupermarketCheckout: The target of the project is a library (.lib), containing the core backend logic of the supermarket checkout, that can be used to feed the input items chosen and produces a receipt.
- SupermarketCheckout_CommandLineInterface: The target of the project is an executable to be launched to use the system through the command line. This executable statically links to the core library in order to interact with it.
- SupermarketCheckout_UnitTest: The target of the project is an executable that launches unit test (via the google test framework GTest), containing the unit test of the core library.

Details of the core backend library

High level class hierarchy:

- Supermarket: The entry point of the library, this class contains the list of products to be bought, allows the customer to checkout a list of items and receive a receipt, the history of receipt is also stored in the supermarket. Also exposes a method to allow the filling of the products in the supermarket.
- Product: A product is a simple structure containing the product name, the unit price, and whether the product is eligible for discounts or not.
- ProductsCount: A simple type definition for an unordered_map of Products to count, marking a group of items of products. It is the main way to group the products in the overall library.
- ProductCatalog: The product catalog is a map of products to the number of items of that product, like a dictionary owned by the supermarket to list the products it contains.
- ProductCatalogFiller: An interface that exposes a product catalog to be filled.
- DiscountManager: The main entry point of the discounting logic, for now simply filters out the products eligible for discount and delegates to the discount strategy to compute the discounted items result.
- DiscountStrategy: An interface that applies the discount on a list of products and returns the discounted items. The implemented strategies:
    - DiscountIdenticalItem: Discount strategy to compute identical item discounting rule
    - DiscountItemCount: Discount strategy to compute the item count discounting rule.
    - DiscountComposite: A list of discounting strategies to apply on the same set of products, also ensures that the strategies are applied in order.
    - DiscountStrategyType: Enum denoting the priority of appliance of discounting rules.
- Receipt: The output containing all the receipt items for non-discounted products, discounted products, and allow to compute total and subtotals.
- ReceiptItem: A simple structure to display the receipt information per product.

High level UML Class Diagram:



Main use case flow:

- Create a Supermarket object while providing the DiscountingStrategyType and threshold values to initialize the DiscountManager.
- After implementing the ProductCatalogFIller, you will be able to fill the supermarket catalog.
- Then you can proceed to check out with a list of items. During checkout, the DiscountManager applies its discounting rules through the DiscountStrategy interface, resulting in a ProductsCount containing the discounted (free) items.
- Chosen items are removed from the ProuctCatalog in the Supermarket.
- Then both the bought the initial list of items and discounted items are fed into the Receipt which is stored in the supermarket as well as returned.

Details of the command line interface

The project that allows to manually test the supermarket checkout library. It has 3 main modes of operations:

- Default Values: This is a simple hardcoded way to interact with the system, both the Supermarket initialization is hardcoded, as well the customer order, resulting in the same behavior each time. This is a for fast and easy use to quickly show the behavior of the system.
- Manual: You have the possibility to input the information you want. Create your products and the count that will be present in the Supermarket, then allows you to choose items to be checked out, and receiving a Receipt. The supermarket default values can also be created in this case for simplicity.
- File: Similar to the Manual Input, however the input mode is through a file input following the same formatting.

Usage flow:

1. First chose the interaction mode.
2. Decide how to fill the Supermarket catalog (If not choosing default mode before).
3. Product Catalog is print to std::cout with each item having an id.
4. Decide if you want to buy items or not.
    a. If you chose not to buy items, go to 5.
    b. Chose the products to buy using the product id along with their count.
    c. Print the chosen products
    d. Decide to checkout or not.
    e. If you checkout the Receipt will be printed out.
    f. Go back to 4.
5. Program Exit.

Input Format: For both the Manual and the File

- To fill the Supermarket catalog: (Example in SupermarketProductCatalogInput.txt)
    - First line should contain the number of products to add.
    - Then a line for each product to add following this format (Separated by spaces):
        - ProductName(Single word)
        - UnitPrice
        - EligibleForDiscount(Y/N)
        - Count
- To fill the customer order: (Example in CustomerOrderInput.txt)
    - First line should contain the number of products to buy
    - Then a line for each product to add to the list (Separated by spaced)
        - ProductId (Taken from the print out of the ProductCatalog)
        - Count

NB: The command line interface does not yet check for inconsistent input and is not safe of usage if not used as expected. So, you might have undefined behavior if inputting inconsistent data. This is of course not ideal, however for the sake of the problem, the simple interface seemed enough and in real life would not really be the real way of interacting with the system.

Details of the unit test library

Unit tests to make sure non regression during development and potential later enhancement, as well as making sure that each component behaves as expected on its own.

Main unit tests were done on the DiscountStrategy part where the main complexity resides. (DiscountStrategyTest.cpp).

SupermarketTest.cpp is more of a component test, testing the whole flow from entry to exit of the library.

Next Steps and Potential Enhancement:

- Protection on the command line interface for inconsistent input.
- A graphical user interface (using Qt) can be created to have a more friendly and safer way of using the system.
- Input and output via XML or JSON format.
- DiscountManager to contain a map of DiscoutingStrategyType to Product to have more flexibility on which product applies to which rule.