Advanced Software Engineering (F21AS): Group 4

Stage 1: Group Report

Dominic Taylor (H00246129)

Aina Centelles Tarrés (H00358958)

Leah Wright (H00360682)

Bo Han Jhan (H00317537)

Jiancheng Zhang (H00341619)

Table of Contents

1. Members' responsibilities

Task	Responsibility
Create menu, create classes for orders, menu & customers	Aina, Jiancheng
Process input files of menus & orders	Bo Han, Jiancheng
Deal with discount rules	Jiancheng
Add exception handling	Dominic, Leah
Add Junit tests	Aina
Add GUI	Bo Han, Jiancheng
GUI functionalities	Bo Han, Jiancheng
S/W integration	Jiancheng
Use Case + Activity Diagrams	Dominic
Class Diagram	Jiancheng
Group report	All

2. Link to Repository

Clone with SSH: **git@gitlab-student.macs.hw.ac.uk:ase_coursework_group-4/coursework.git**

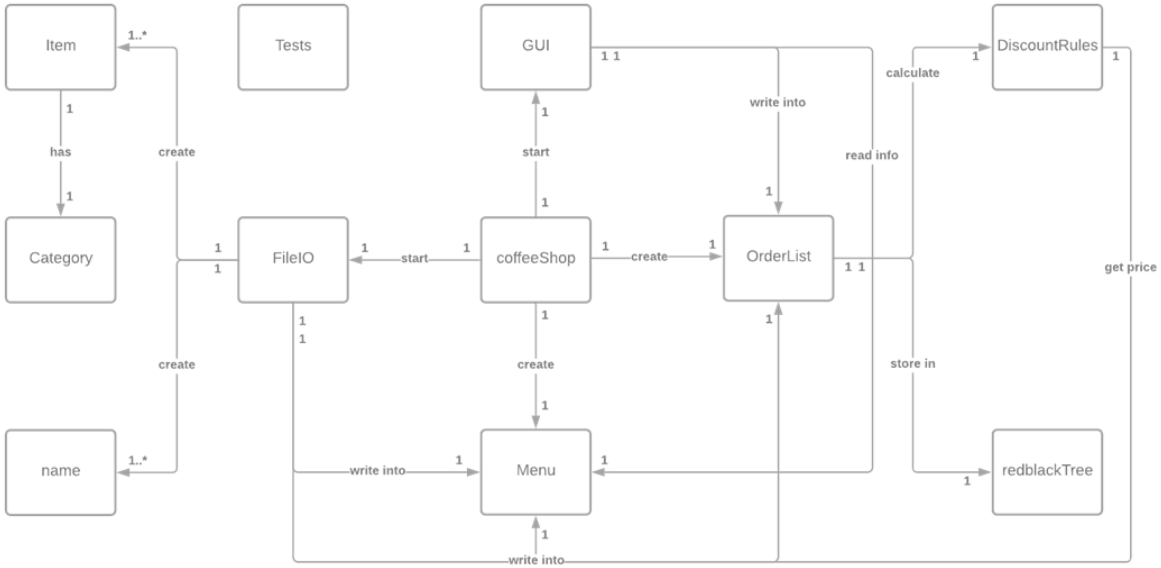
3. Program Specification

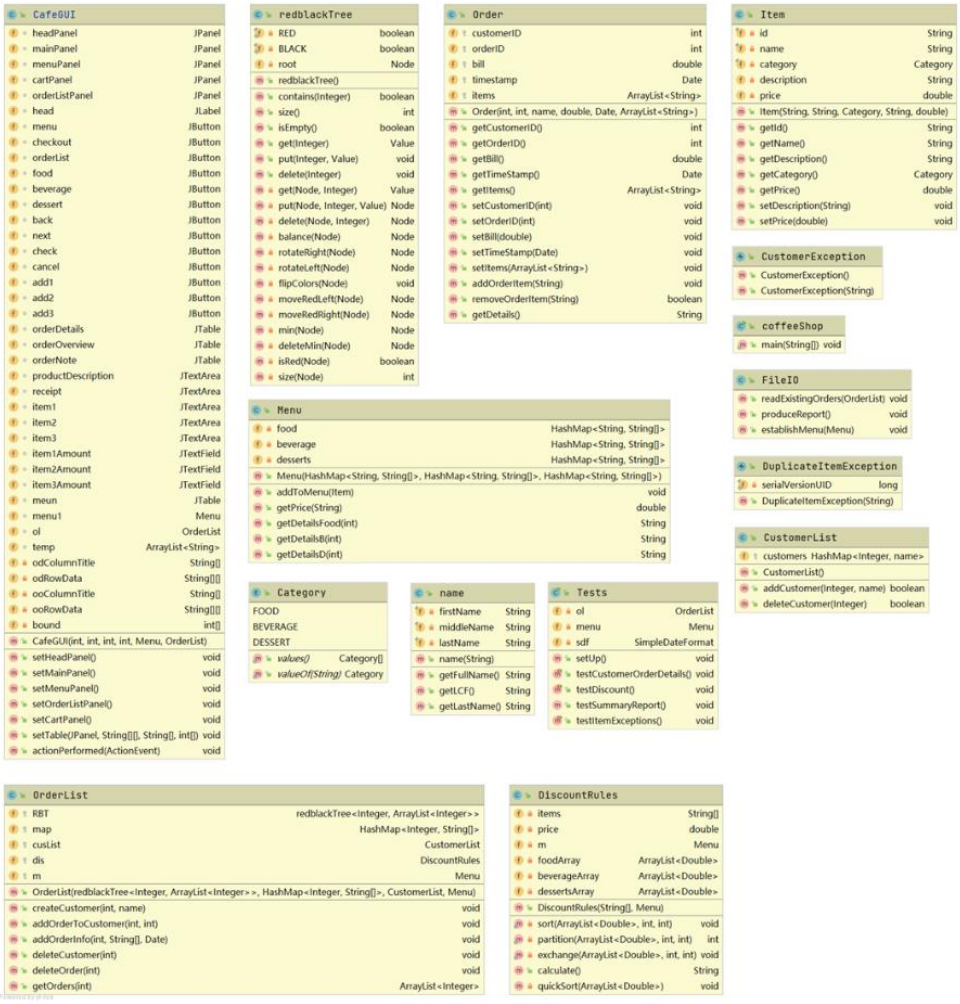
Basically, **this program meets all specification** this coursework requested. The text file, ExistingOrders.txt, stores a list of existing customer orders, which allows a customer number to appear with ordering multiple items. There is another text file, Menu.txt, providing details of all categories of product, which include each item's unique identifier and brief description. After end the application, a summary report generated contains statistics for orders.

There are few problems found in the development although they won't affect the basic functions of the system. For example, the UI designer organized several elements in a panel in order to avoid creating too many windows when using the application for ordering. Therefore, the users can take a few steps on completing their order. The buttons located in the main panel were assigned to control what should be showed in the central penal. It seems working improperly when clicking buttons in the main panel, the central panel won't transform to display a specific content until the user manually adjusts the size of the window. The developers have no idea why the application perform in this way but they develop the application logically right and meet the requirements in accordance with the coursework description.

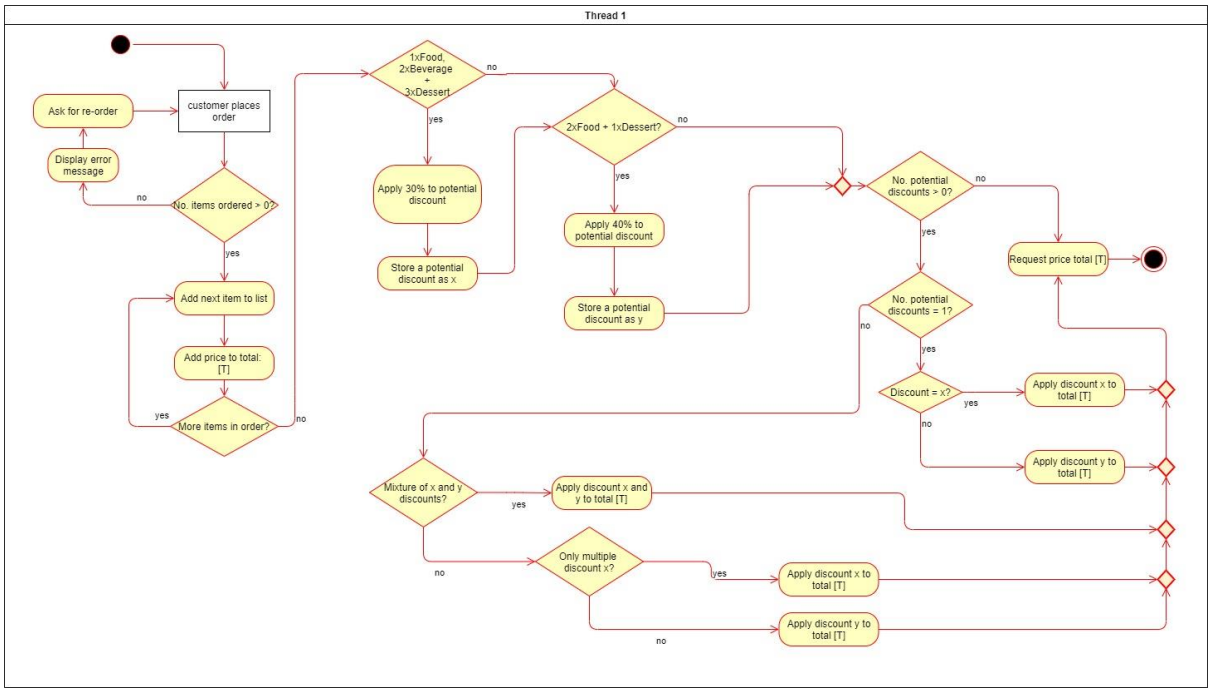
4. UML Diagrams

Class diagram

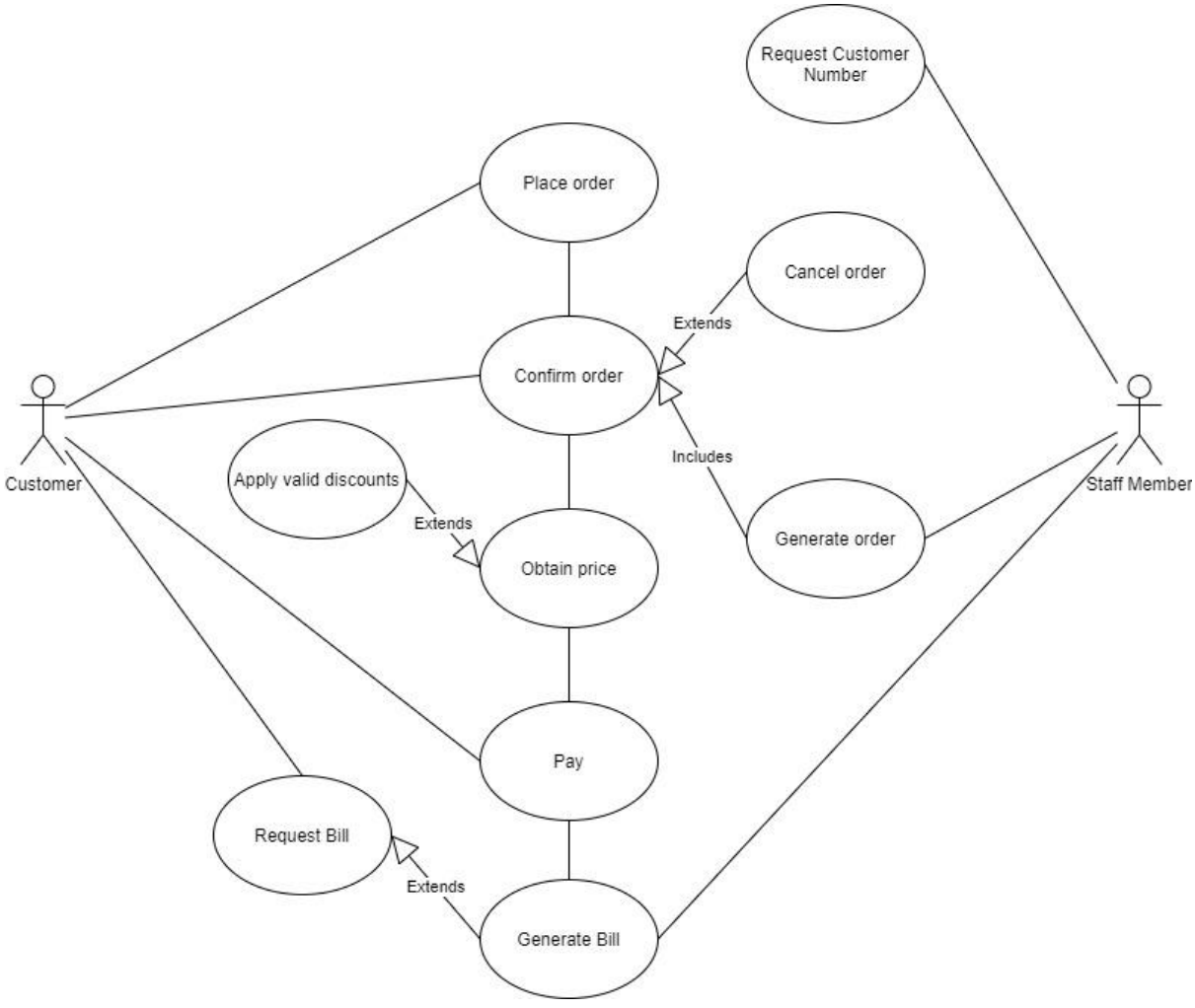




Activity diagram



Use Case diagram



5. Data structures

When the number of customers and orders is larger, the probability that the same customer ID corresponds to the same hash code is relatively higher, which means it is highly possible for hash code collision to occur. In addition, when the length of the HashMap is greater than 64 and more than 8 elements are stored, it will be stored in the form of a **red-black tree**. Assuming that when the number of customers and orders is large, the red-black tree is directly used as the data structure, and the time complexity of insertion, deletion, and search are all O(log(n)). As for each order number, it only corresponds to a specific order. At this time, it is directly stored in a **HashMap**, and the search time complexity is only O(1). In the same way, each group of customer ID will only correspond to a customer name, and each product ID will only correspond to a specific product, so data has similar relationship between themselves has been determined to be stored in **HashMap**.

6. Functionality decisions

Identifiers:

- Item:
    - **itemID**: can be form FOOD001, BEVERAGE001, DESSERT001 i.e. category followed by three numbers, it will be unique to each item. This is in the form of CATEGORY001 (the category the item belongs to in capital letters, followed by three integers beginning with 001 and increasing incrementally)
- Order:
    - **custID**: this is an id of the customer who is associated with the order therefore if a customer makes multiple orders, the **custID** will be the same for each order. It is in the form of 10001 (5 integers), it begins with 10001 and increases incrementally as new customers are added.
    - **orderID**: this is the ID of the order, it is unique to each order therefore if a customer makes multiple orders, each of their orders will have a different **orderID**. It is in the form of 101 (3 integers) beginning with 101 for the first order and increasing incrementally for each new order.

Discounts:

- 30% reduction if 1xfood, 2xbeverage and 3xdessert
- 40% reduction if 2xfood and 1xdessert

A discount can only be applied once to each order, however if it is possible to have both discounts applied, they will be.

7. Testing

Three JUnit tests are included in the "tests" package:

I. **testCustomerOrderDetails()** checks that multiple orders can be added to the same customer and successfully retrieved using the method **OrderList.getOrders(int custNo)**. Testing cases include customers with multiple orders and customers with no existing orders.

II. **testDiscount()** checks that discounts are properly applied to orders by the method **DiscountRules.calculate()**. Test cases are:

- Only one discount can be applied.
- Both discounts could be applied so the method must choose the best one for the customer.
- No discounts can be applied.
- Both discounts should be applied to the same order so the method must choose which discount should be applied to which item.
- One discount can be applied twice but should only be applied once (a specific discount can only be applied once per order) to the items that maximise the customer's benefit.

III. **testItemExceptions()** checks that exceptions are thrown for the creation of incorrect items. More specifically, it checks that

- An **IllegalStateException** is thrown by the Item constructor when an Item is created with no ID, no name or no description.
- An **IllegalArgumentException** is thrown by the Item constructor when an Item is created with negative or price 0.
- A **DuplicateItemException** is thrown by the method **Menu.add(item)** when an item is added to the menu with an already existing itemID.