Advanced Software Engineering

Group Report 1

[Calum Henning, Duncan Austin, Katarina Alexander, Kate-Lynne Thomson & Stuart Paterson]

Link to GitHub: https://github.com/chenning17/Advanced_Software_Engineering

Summary of work completed

The work was split evenly according to ability and previous experience. Each person in the group had their own parts to work on, as seen below:

Calum: Created following classes: Item, Snack, Food, Drink, Discount, DuplicateIDException, InvalidIDException. Worked on DiscountCalculator class, tests for Item (and subclasses).

Duncan: Created tests for Item subclasses, implemented GUI and majority of its simple functionality.

Katarina: Created the FileInput, OrderLoader and ItemLoader classes, the menu.csv and DiscountCalculator tests. Worked on the output report in the Manager class.

Kate-Lynne: Created itemCollection and orderCollection classes, and the orderLoader and itemLoader tests. Worked on the order.csv and the output report in the Manager class.

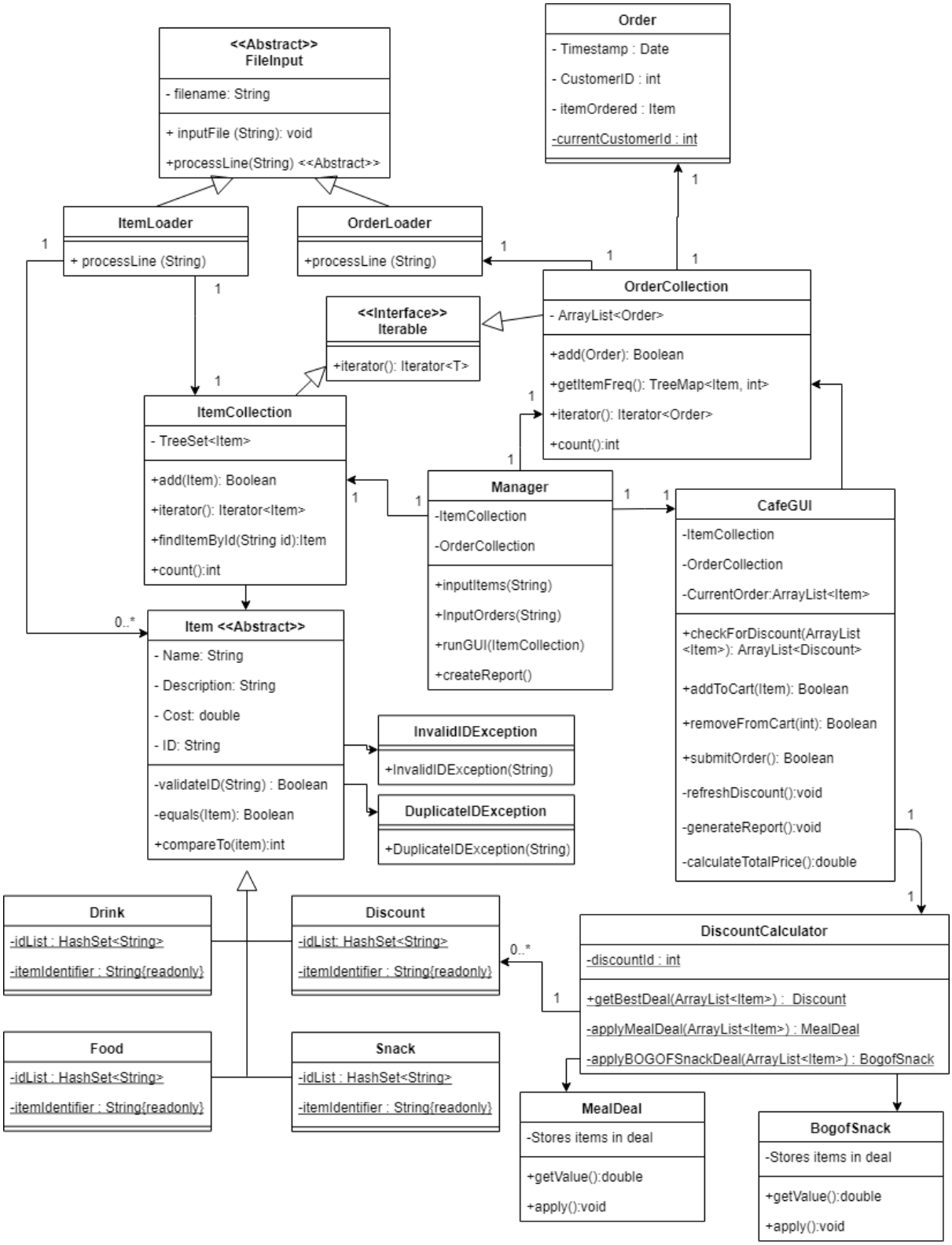
Stuart: Order unit tests, Order class, early report generation, expanded discount functionality to allow multiple discounts to be added, added to; manager, GUI, collection classes. General bug fixing, testing and refactoring.

All: Both the Development Plan and Group Report were created by everybody collaboratively and equally.

The Programme Specification

This program meets the specification fully.

UML Class Diagram



Data Structures

There were different forms of collections used within this program. These were chosen dependent upon what was needed from the program and how we wanted the data to be held in the data structure.

HashSets:

Each subclass of the Item class has a static hashSet containing all the id strings for items of that category. When a new Item subclass is instantiated the id string passed to the constructor can be checked against this set. If it already exists a DuplicateIDException will be thrown because two items cannot have the same id string. This exception is caught in the add method of the ItemList and this method will then return false to show the add failed.

TreeMap:

Used in OrderCollection class, in the getItemFreq method. The map is an easy way to associate menu items (used as keys) with a count of how many of them were sold in that session of the report (integer values).

TreeSet:

Used to store menu items in the ItemCollection class. This is an appropriate choice as the majority of the tree is created at the beginning of the program as items are loaded from the CSV file. Additions to the tree only happen occasionally when a new discount item is created. This is to accommodate report generation, as all items that need to be displayed in the report are kept in the same place. While there is a performance trade-off for addition to a tree and potentially rebalancing it, there is only a finite number of discounts that can be added. The tree allows items to be sorted alphabetically for both display in the GUI and in the report generated at the end.

ArrayList:

Used when operations on the data structure are largely accessing its elements in sequence. Stores orders in the OrderCollection, which is useful for iterating over the data structure to get the frequency of items sold. Also used in the GUI for passing subclasses of items to the DiscountCalculator and returning all applicable discounts to be displayed in the GUI.

Programme Functionality

The decision was made to represent anything that could appear in the menu or report as an extension of an abstract Item class. The 3 categories of item a customer can order are Food, Snack and Drink. Identifiers for these are comprised of 4 lowercase characters that signify each item category, followed by three numbers that are used to ensure uniqueness.

Discounts have also been modelled as subclasses of the Item class. This allows them to be grouped into the cart or used to create orders like other Items. Discounts can then be subtracted from the cart total cost.

Discounts are applied to the cart using a “best first” approach - discounts with the highest monetary value are applied first. More than one discount can be added to the cart if applicable, but any item may only be part of one discount. *Figure 1* demonstrates how this is achieved.

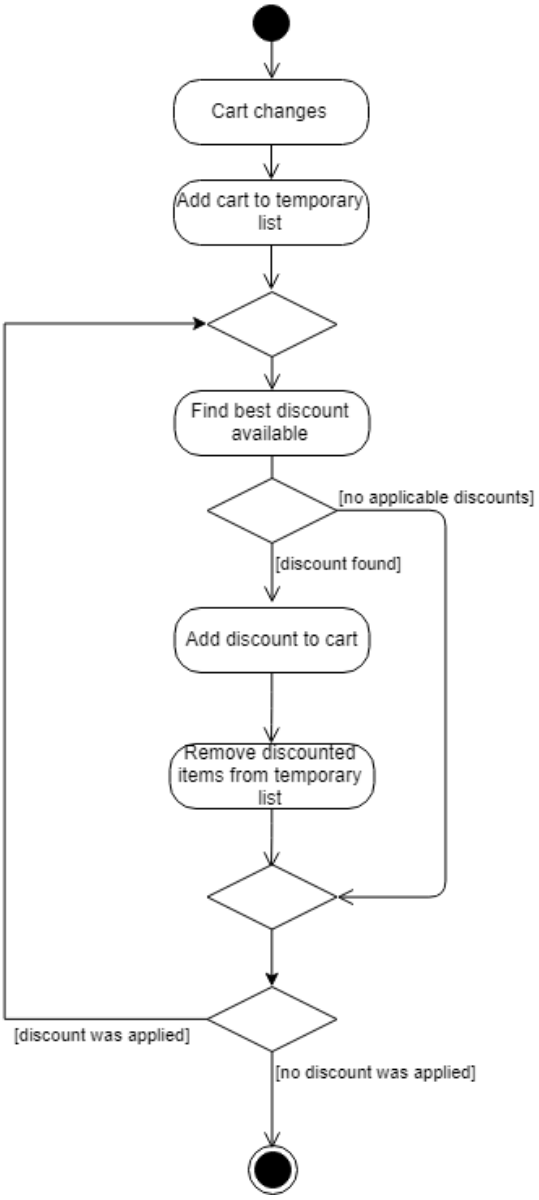


Figure 1

Every time the cart is updated discounts are recalculated to ensure they are always up to date.

The two discounts implemented are shown in *table 1* below:

Discount	**Details**	**Conditions**
Meal Deal	One food, one snack, one drink for £5.50.	Total price is greater than 5.50.
Snack buy one get one free	Any two snacks, cheapest one free.	

Table 1

Orders are modelled essentially the way they are described in the CSV file, with a single item associated with a customer ID and timestamp, rather than being grouped. This was originally done due to assuming that the format of the order CSV needed to be the format of the order object. As one of the functional requirements of this stage of the coursework is planned iterative development, and the software engineering requirements state that all planning should be done before starting to write code, the decision was made to follow our plan.

The GUI has basic functionality to allow the addition of new orders. Items are displayed on the left of the screen, controls are in the centre, and items that are being added to the new order are on the right. Multiple items can be selected (by control-clicking) and added or removed at the same time. Each time the list on the right changes, any applicable discounts are automatically added. Creating an order clears the current selection of items, resets the total price, and adds an Order for each item, using the same customer ID and timestamp. A screenshot of the GUI (*Figure2*), and activity diagram (*Figure3*) of its use are provided below:

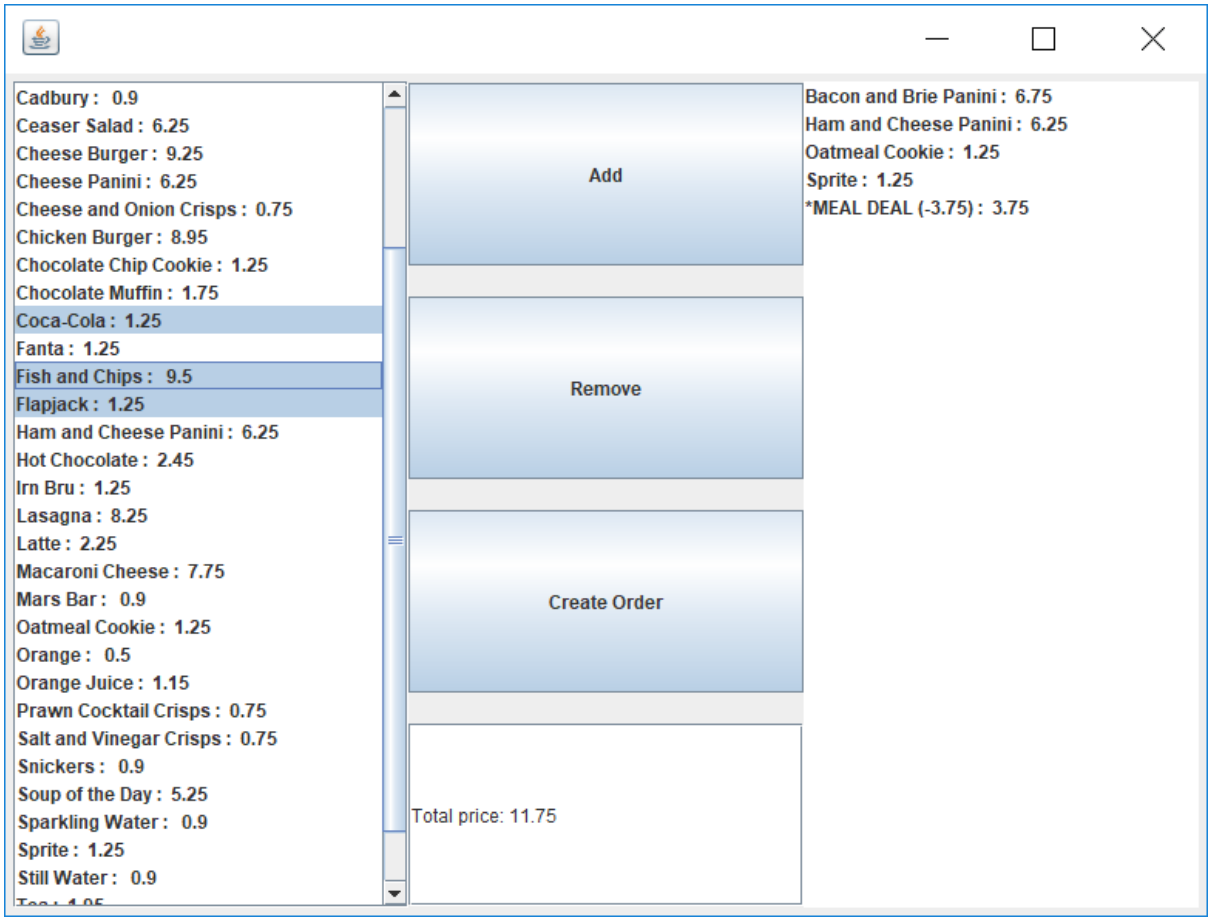


Figure 2

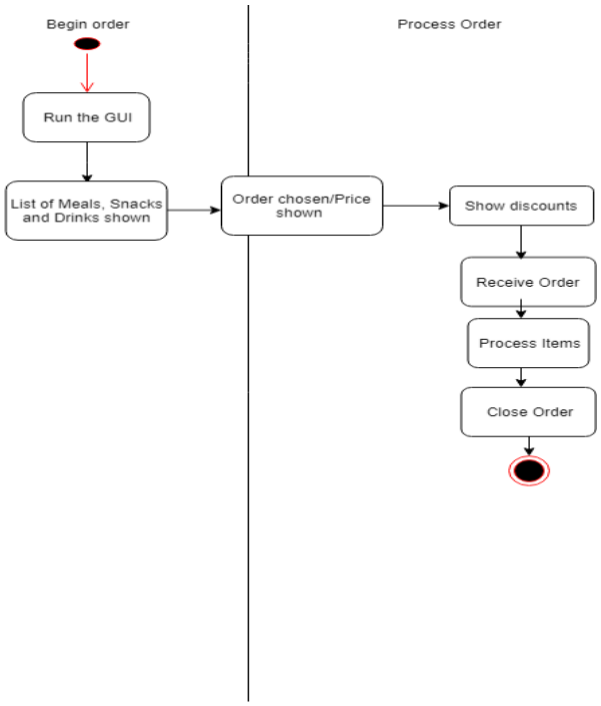


Figure 3

Reports are automatically generated when the GUI is closed, as shown in *figure 4*, and provide information about how many of each item were sold, how many discounts were applied, and the gross profit for the day. The report is named after the day it was created. Items in the report are grouped based on their subclass, and then shown alphabetically.

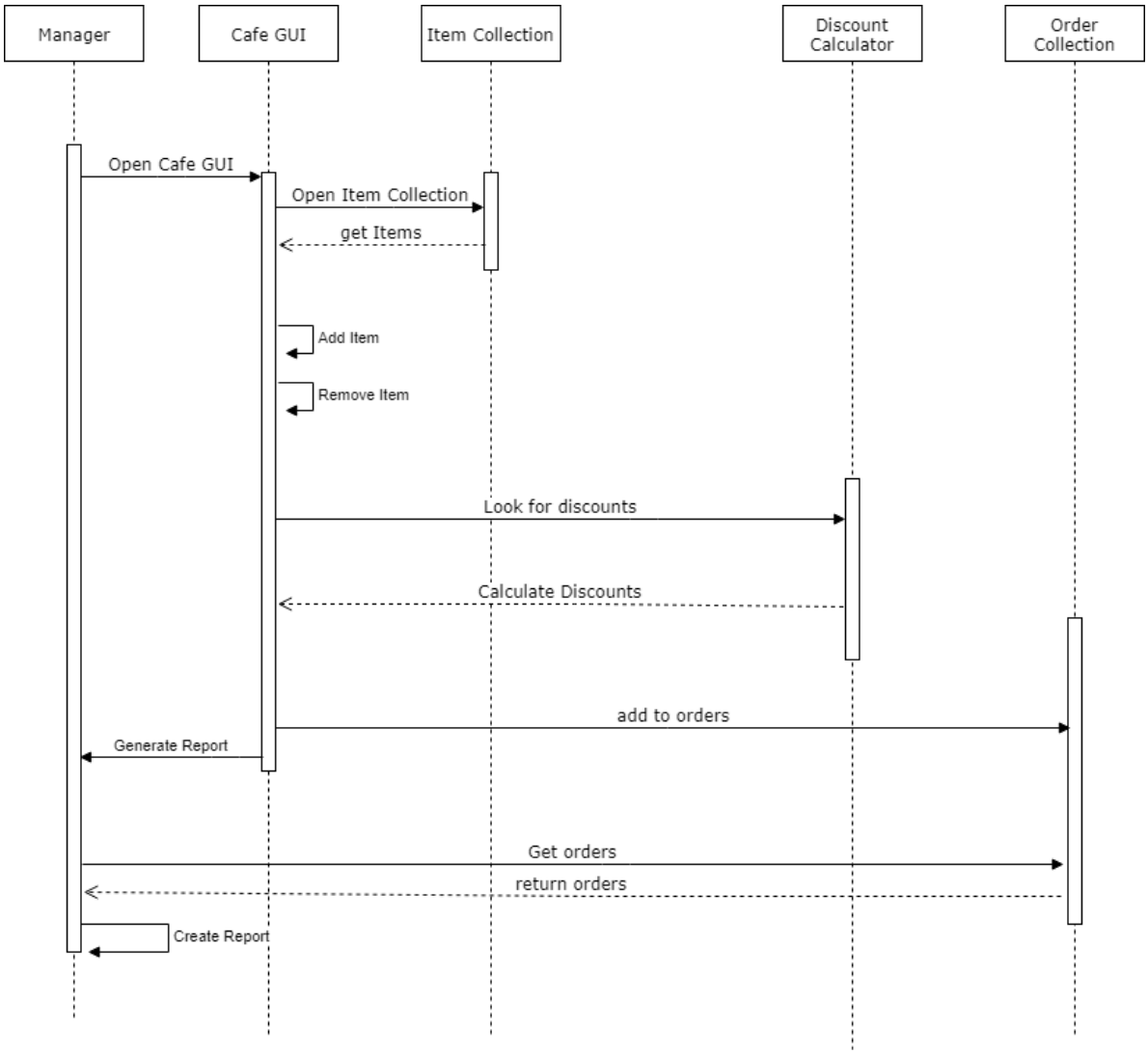


Figure 1

Testing

Item subclass Tests (Food, Drink, Snack, Discount):

These tests cover the getter and setter methods and test the constructor in case of invalid ID or when a negative item price is used.

Order Tests:

Tests that the orders are created with dates in the expected format, and that customer IDs and Items are all correct. Also covers the constructor using Null instead of various values, which throws an IllegalArgumentException.

DiscountCalculator Tests:

Tests that the correct discount types and reductions are being applied when given an ArrayList of orders. The tests cover single and multiple instances of discountable items; ensure that the Meal Deal discount is not applied if the total cost is under £5.50; check that the BOGOF Deal only takes off the cost of the cheapest item if three snacks selected.

OrderLoader/ItemLoaderTests:

Tests the reading in of CSV files by writing in a known and correctly formatted string to a one-line CSV file, then attempting to read that line back in and create the object correctly.

Exceptions:

We created two new exception classes, InvalidIDException and DuplicateIDException. InvalidIDException is raised if an attempt to initialise an Item with an incorrectly formatted ID is made. This is thrown when reading from the CSV file if the program doesn't recognise which class constructor to call, or by the constructors themselves if the ID does not adhere to the specified format. Each Item category contains a shared static HashSet of IDs used to check if an ID already exists for a given Item type. Therefore, if there is an attempt to initialise an Item when an ID already exists a DuplicateIDException is raised by the constructor.

We also handled the FileNotFound exception in FileInput, IOException in the report generation, Exceptions in the Manager and OrderLoader classes.