Group 7: Mirko La Porta, Odhran D McHugh, Karan Singh, Emanuele De Pellegrin, Krishna B Nair

Group Development Plan

This document shows the development plan of a Java application that will simulate a coffee shop. This document is the very first step towards the creation of the application and was realised before writing the code. The following sections explain in the detail what choices were made and their related motivations. In particular:

- Section 1 describes behaviour and structure of the application using three UML diagrams: Use Case, Activity and Class Diagrams.
- Section 2 lists the data structures of the application explaining why they were chosen.
- Section 3 describes the work plan through a Gantt Chart Diagram, in which it is possible to see that planned iterative development will be applied during the implementation.

1. Design

1.1 Behaviour

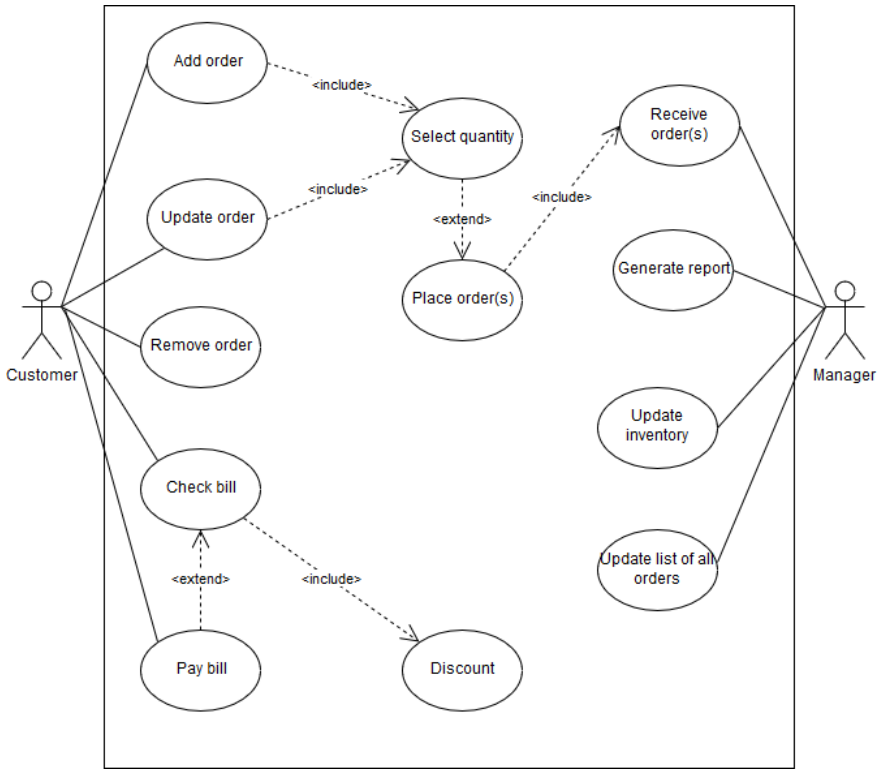

Figure 1: UML Use Case Diagram

Specifying all the possible roles that a user can play while interacting with the system (i.e. the application) as well as all the possible interactions was necessary to determine the behavioural requirements of the application. To do so, a UML Use Case Diagram was realised (*Figure 1*). It is important to highlight that the Manager receives the Customer's confirmed orders, and update the inventory and the list of all orders accordingly. Indeed, the Manager needs to keep a list of all the orders so that he can generate a report right before the application exits.

Two interesting scenarios came out while refining the Diagram. One of them was "the Customer wants to change the selected item of an order", which was a scenario within the Update Order use case. This scenario was not included, since the same purpose can be accomplished by removing the current order and adding a new one through the respective use cases. The other scenario was "the Customer wants to order an item that is not present in the inventory or a greater quantity than the one in the inventory". It was thought that this scenario can be handled through exceptions.

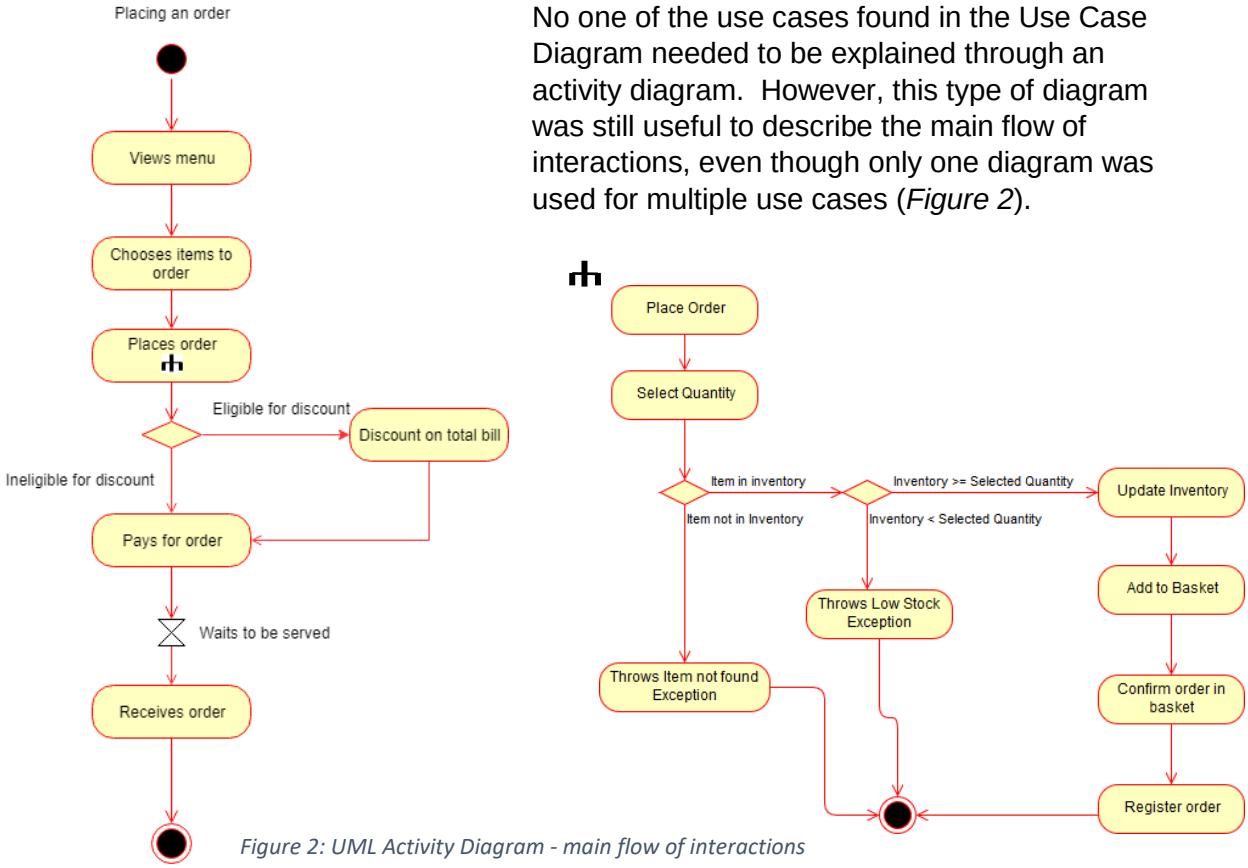

Figure 2: UML Activity Diagram - main flow of interactions

No one of the use cases found in the Use Case Diagram needed to be explained through an activity diagram. However, this type of diagram was still useful to describe the main flow of interactions, even though only one diagram was used for multiple use cases (*Figure 2*).

1.2 Structure

After determining the behavioural requirements of the application, it was necessary to determine also its structure in terms of the attributes and responsibilities of each class as well as the relationships among classes. To do so, a UML Class Diagram was realised (*Figure 3*). The layout of this Diagram aims to show that everything is centralised towards the Manager class, which is also the class that contains the main method. For this reason, the Manager class has three static attributes that are all the entities needed by the other classes. For example, if the Basket method named *checkUpdateInventory* is called, it will not call the Inventory class directly; instead, it will call the Manager class method named *getInventory* and access the static attribute of the Manager class. This Class Diagram shows an important composition relationship and multiplicity between the Food and Order classes: order has exactly one food, while food can have multiple orders[1].

2. Data structures

- Inventory class will use a TreeMap to sort the items. This is preferable because the items will be easier to find. Compared to other data structures, the Tree ones are quicker for searching and this will make updating stocks more efficient.
- Basket class will use a LinkedList to store items ordered by the customer, since it is efficient to add and delete items and these actions will be done more often than accessing items.
- OrdersRegistered class will use ArrayList to store the items ordered, because the items will be accessed sequentially only to generate the final report.

[1] This was necessary since the application specifications specify that multiple items from the same customer must appear as different orders.

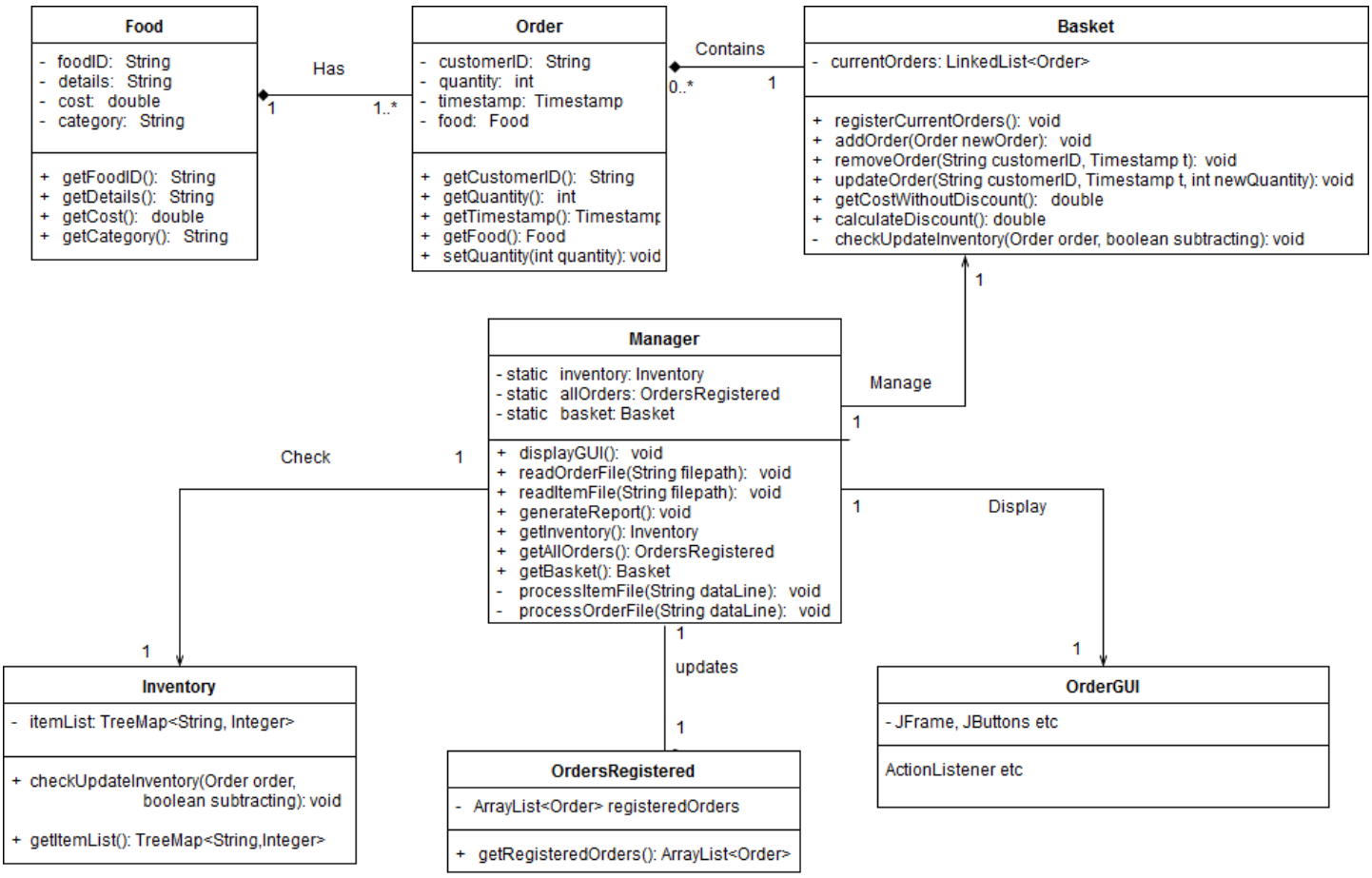

Figure 3: UML Class Diagram

3. Work plan

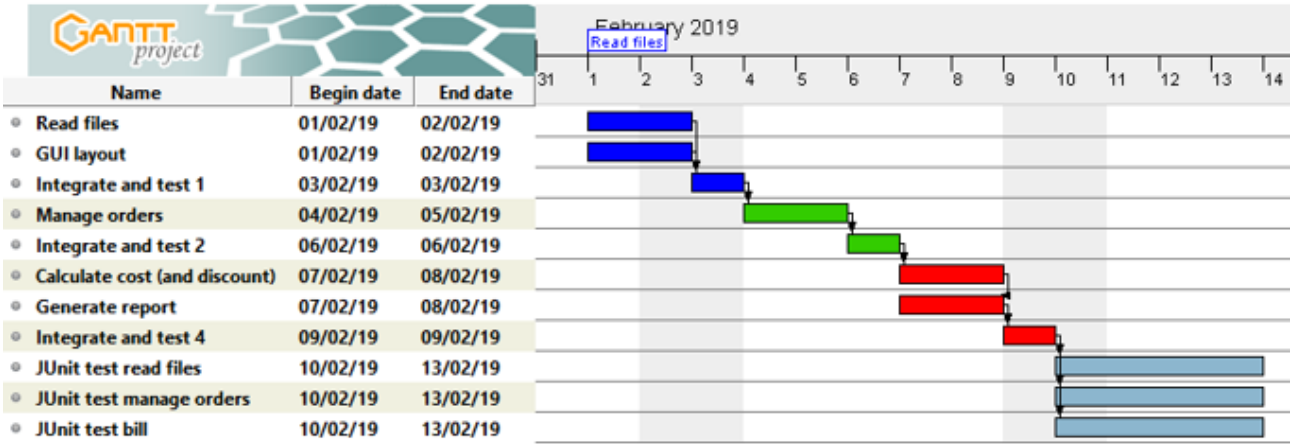

Figure 4: Gantt Chart for the work plan

A Gantt Chart (*Figure 4*) was realised to better show the tasks of each iteration, since the implementation will be carried out using planned iterative development. There will be four iterations: the application will be developed during the first three iterations while the last one will be used for JUnit testing. It is clear an attempt to give a similar amount of work and time to each iteration while avoiding getting stuck waiting for a task to be completed. This happens obviously for each *integrate and test* task and only one more time, since *generate report* task needs *calculate cost* in order to be completed.