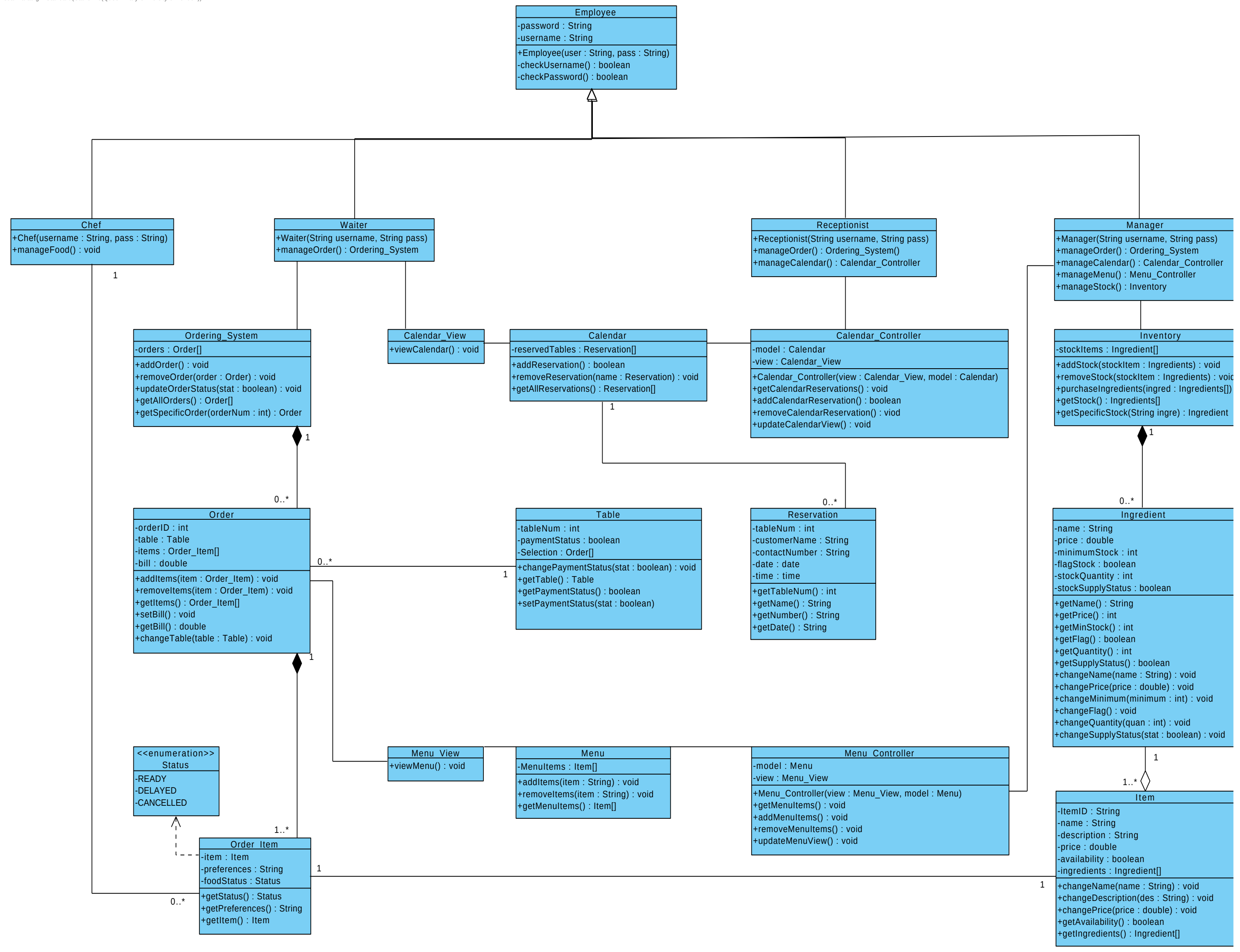
Group 27
Fudo Hyoka
ECS506U Software Engineering Group Project
Design Report

1. Class Diagram (40%)


Employee
-password : String
-username : String
+Employee(user : String, pass : String)
-checkUsername() : boolean
-checkPassword() : boolean
Chef
+Chef(username : String, pass : String)
+manageFood() : void
Waiter
+Waiter(String username, String pass)
+manageOrder() : Ordering_System
Receptionist
+Receptionist(String username, String pass)
+manageOrder() : Ordering_System()
+manageCalendar() : Calendar_Controller
Manager
+Manager(String username, String pass)
+manageOrder() : Ordering_System
+manageCalendar() : Calendar_Controller
+manageMenu() : Menu_Controller
+manageStock() : Inventory
Ordering_System
-orders : Order[]
+addOrder() : void
+removeOrder(order : Order) : void
+updateOrderStatus(stat : boolean) : void
+getAllOrders() : Order[]
+getSpecificOrder(orderNum : int) : Order
Calendar_View
+viewCalendar() : void
Calendar
-reservedTables : Reservation[]
+addReservation() : boolean
+removeReservation(name : Reservation) : void
+getAllReservations() : Reservation[]
Calendar_Controller
-model : Calendar
-view : Calendar_View
+Calendar_Controller(view : Calendar_View, model : Calendar)
+getCalendarReservations() : void
+addCalendarReservation() : boolean
+removeCalendarReservation() : viod
+updateCalendarView() : void
Inventory
-stockItems : Ingredient[]
+addStock(stockItem : Ingredients) : void
+removeStock(stockItem : Ingredients) : voi
+purchaseIngredients(ingred : Ingredients[])
+getStock() : Ingredients[]
+getSpecificStock(String ingre) : Ingredient
1
1
0..*
1
0..*
1
0..*
Order
-orderID : int
-table : Table
-items : Order_Item[]
-bill : double
+addItems(item : Order_Item) : void
+removeItems(item : Order_Item) : void
+getItems() : Order_Item[]
+setBill() : void
+getBill() : double
+changeTable(table : Table) : void
0..*
1
Table
-tableNum : int
-paymentStatus : boolean
-Selection : Order[]
+changePaymentStatus(stat : boolean) : void
+getTable() : Table
+getPaymentStatus() : boolean
+setPaymentStatus(stat : boolean)
Reservation
-tableNum : int
-customerName : String
-contactNumber : String
-date : date
-time : time
+getTableNum() : int
+getName() : String
+getNumber() : String
+getDate() : String
0..*
Ingredient
-name : String
-price : double
-minimumStock : int
-flagStock : boolean
-stockQuantity : int
-stockSupplyStatus : boolean
+getName() : String
+getPrice() : int
+getMinStock() : int
+getFlag() : boolean
+getQuantity() : int
+getSupplyStatus() : boolean
+changeName(name : String) : void
+changePrice(price : double) : void
+changeMinimum(minimum : int) : void
+changeFlag() : void
+changeQuantity(quan : int) : void
+changeSupplyStatus(stat : boolean) : void
1
<<enumeration>>
Status
-READY
-DELAYED
-CANCELLED
Menu_View
+viewMenu() : void
Menu
-MenuItems : Item[]
+addItems(item : String) : void
+removeItems(item : String) : void
+getMenuItems() : Item[]
Menu_Controller
-model : Menu
-view : Menu_View
+Menu_Controller(view : Menu_View, model : Menu)
+getMenuItems() : void
+addMenuItems() : void
+removeMenuItems() : void
+updateMenuView() : void
1
1..*
Item
-ItemID : String
-name : String
-description : String
-price : double
-availability : boolean
-ingredients : Ingredient[]
+changeName(name : String) : void
+changeDescription(des : String) : void
+changePrice(price : double) : void
+getAvailability() : boolean
+getIngredients() : Ingredient[]
1..*
Order_Item
-item : Item
-preferences : String
-foodStatus : Status
+getStatus() : Status
+getPreferences() : String
+getItem() : Item
1
1
0..*

2. Traceability matrix (10%)

Class	**Requirement**	**Brief Explanation**
Employee	General - RQ2	Contains an internal database of users stored.
Employee	General - RQ4	Every User has their designated class that contains a username attribute.
Employee	General - RQ6	The database accessed within Employee is used via the methods: checkUsername and checkPassword, which verify from the database that the user is valid and exists in the system.
Ordering_System	Waiter - RQ2	The class interacts with the Orders database internally, when a new Order is added a new entry is added to the database.
Table	Waiter - RQ9	The table class has a payment status attribute that can be manipulated via the operation – changePaymentStatus(). The status can then be retrieved by another getter method.
Item	Waiter - RQ12	The item class represents a specific dish and drinks available from the menu, the attribute availability is used to record whether the item is available to the customer.
Calendar	Receptionist – RQ2	In the Calendar class, an array of reservations are held, this is all internally stored in a database, when a new reservation is added this is added to the database.
Reservation	Receptionist – RQ3	The Reservation class contain all the crucial attributes for a table reservation.
Inventory	Manager – RQ3	The inventory acts as a database, internally manipulating it and accessing it when need for information about the stock.
Menu	Manager – RQ4	The menu class has an array of Item type objects which contain all of the information about the menu item.
Item	Manager – RQ5	The Item class has an array of Ingredients from the Inventory.
Ingredient	Manager – RQ6	Each Ingredient in the Inventory class have the attribute – stockSupplyStatus, which hold an enum.

3. Design Discussion (20%)

1. Design choices: Our original design used inheritance to model the way in which different actors had access to different functionality of the system. For example, a Receptionist was considered a subtype of Waiter, as a receptionist would perform an extended set of the Waiter's duties. However, this was not intuitively understandable due to the lack of an "is a" relationship. While making the overall design simpler via polymorphism, this choice did not capture the true nature of how each user interact with the system in the domain. In contrast, the newer design holds all operations a user can undertake in a class specific to that user type, providing cleaner code when needing to access specific methods from a particular user.
   Classes such as Ordering_System and Inventory, which are accessed by the various subtypes of Employee, implement the use case logic and act as a gateway to the classes lower down, hiding their complexity.
   Excluded attributes: The Table class does not contain an attribute to hold all of its reservations because it is more efficient to hold the information about reservations for tables in a centralized calendar. Certain attributes implied by associations lines have not been explicitly listed.

2. Associations: Each user is represented as an object of their own, containing methods which demonstrate the behaviours each user can undertake. The crucial behaviours they can undertake are shown with association lines, while other behaviours don't have an association line. For example, A receptionist is also able to manage orders, just like a waiter, but this is shown as an operation in the class to avoid many association lines.
   Multiplicities: In general the diagram includes several 1 - 0 ... * multiplicities wherever one entity in the system manages a collection of items, such as a calendar managing a collection of reservations, the inventory managing a collection of ingredients, etc.
   Aggregations: Item is an aggregate of ingredient because an Item e.g. a particular dish, is composed of many ingredients, while an ingredient does exist despite an item not.
   Composition: The Inventory class is a composition of Ingredient items, as an inventory is required to track items. If the inventory is removed, the ingredients belonging to it cannot be tracked and hence it makes sense to remove them. An Order is a composition of Order_Item objects, as an Order_Item is a particular food item being prepared as part of an order. If the order is cancelled the food item is no longer needed and is removed.

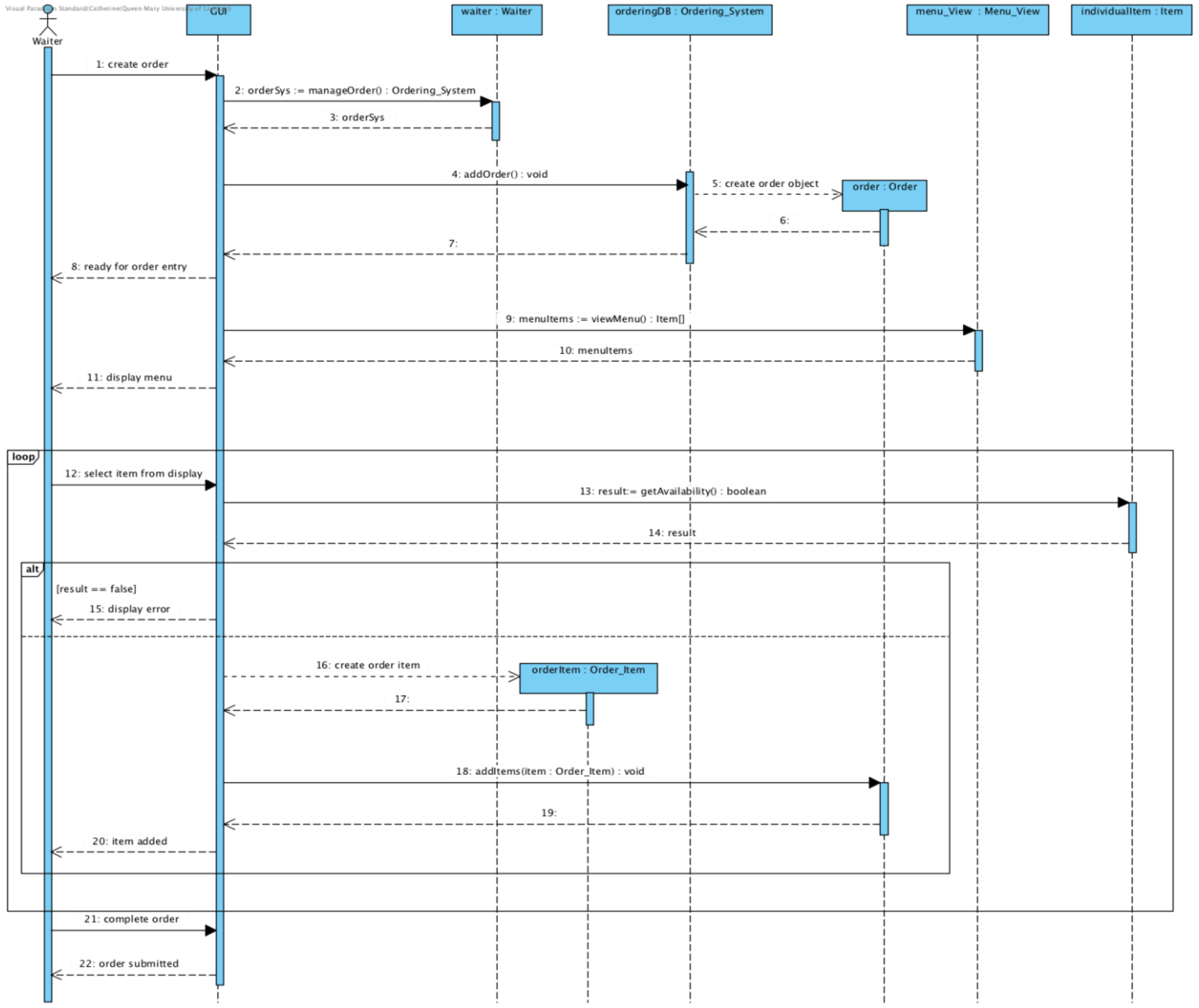
3. The takeaway functionality has been removed as it adds unnecessary complexity to the system by needing to set an additional class to hold orders that are not associated to a table. This however could easily be reinstated during the implementation by adding a pseudo table number for takeaway orders, where the system will recognise it as a 'fake/virtual' table designated only for takeaways.

4. The Model-View Controller (MVC) pattern was applied, as it perfectly accommodates the different roles involved in the system. The following scenario demonstrates the reason for using the pattern.
   Both a Waiter and Receptionist have 'access' to the calendar, but to different extents. A waiter can only view the calendar (All the reservations), while a receptionist both views and manipulates the Calendar, as they add and remove reservations. Without the MVC pattern, all these methods would be compressed into a simple Calendar class, which would be accessed by both Waiters and Receptionist. However, there would be added complexity for checking the 'access privilege' of the user, as Waiters would not be allowed to access methods for adding or removing. The MVC separate these crucial role differences by providing an abstraction (the view) of the Calendar, that the Waiter class does not care about.

4. Sequence Diagram 1 (15%)

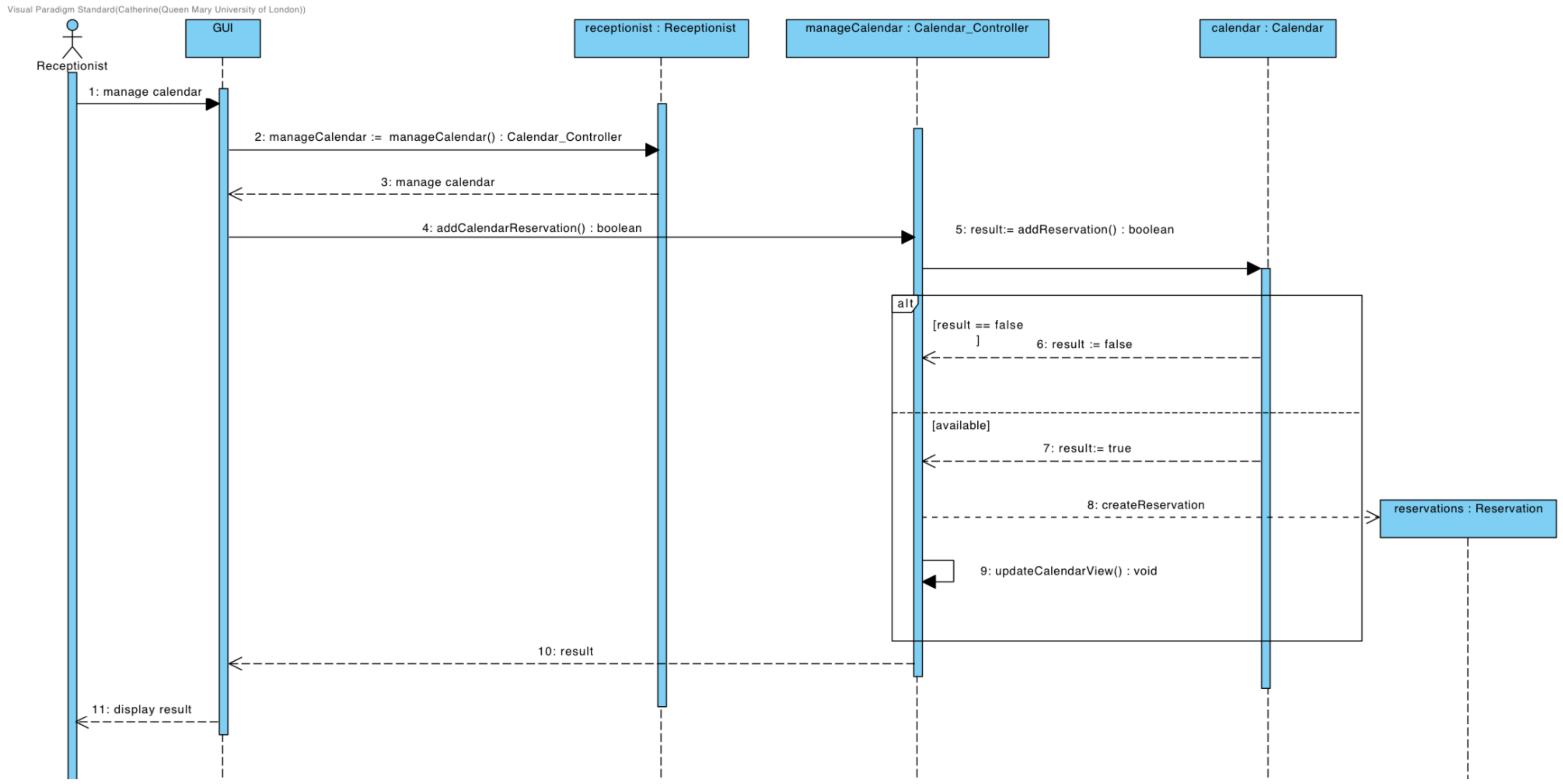
Use case: Manage order

Note: The use case description for Manage order was too complex to add into one sequence diagram, as a result flows such as; food status and change payments, haven't been added. Additionally, certain alternate flows e.g. removing an item is not captured in the sequence diagram. Hence, the following sequence diagram is for adding an order to the system and items iteratively.



Sequence Diagram 2 (15%)

Table reservations



Visual Paradigm Standard(Catherine(Queen Mary University of London))
Receptionist
GUI
receptionist : Receptionist
manageCalendar : Calendar_Controller
calendar : Calendar
1: manage calendar
2: manageCalendar := manageCalendar() : Calendar_Controller
3: manage calendar
4: addCalendarReservation() : boolean
5: result:= addReservation() : boolean
alt
[result == false]
6: result := false
[available]
7: result:= true
8: createReservation
reservations : Reservation
9: updateCalendarView() : void
10: result
11: display result