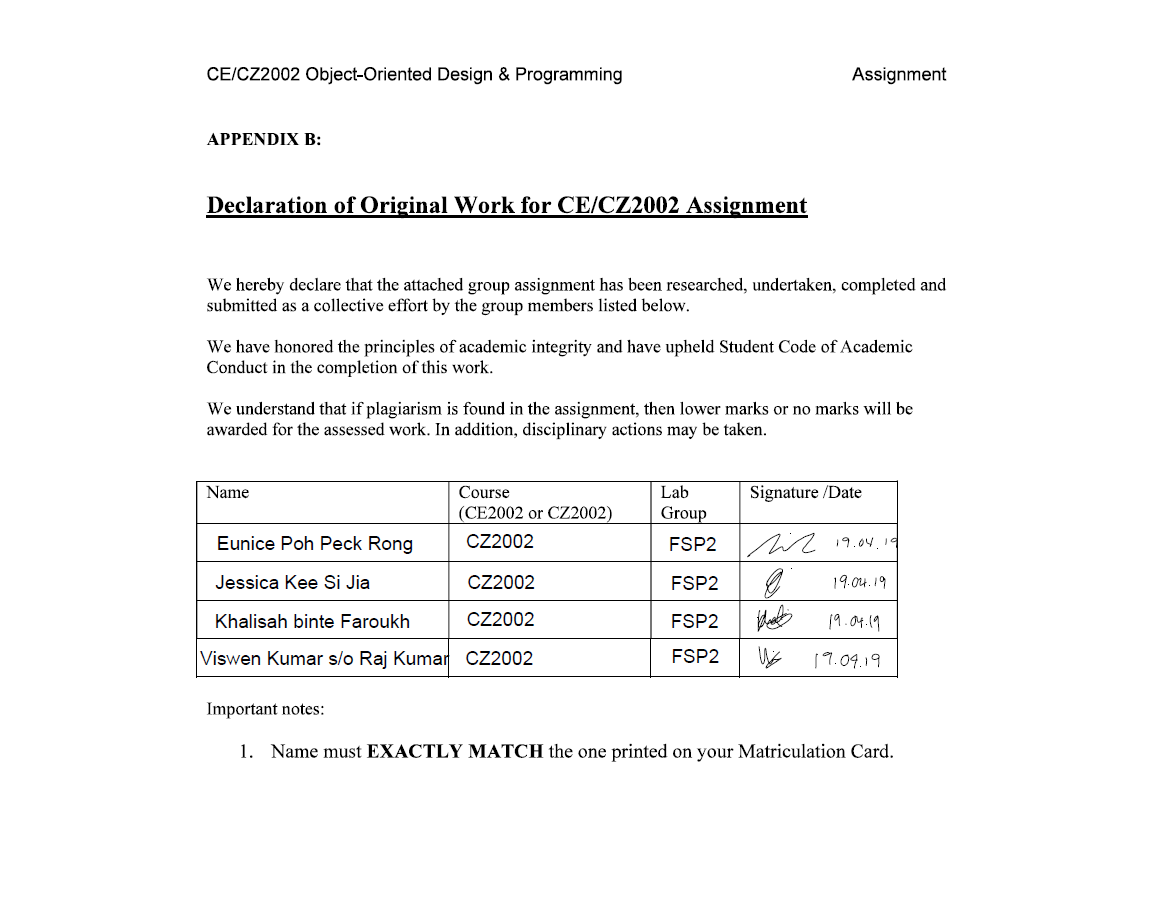


****

**Background**

This report is submitted as part of the CZ2002 Object-Oriented Design and Programming assignment. This report is divided into 5 main sections:

* Section 1 - Design considerations
* Section 2 - Class diagram for the application
* Section 3 - UML Sequence diagram for the “Print Bill Invoice” Function
* Section 4 - Test cases for application
* Section 5 - Consideration for future implementations

The video demonstration and clear image files of the diagrams will also be submitted along with this report.

**Section 1 - Design considerations**

**Approach Taken:**

We have used Data Encapsulation as our object oriented approach for our coding and this was done by setting most of our class attributes to private. Each class has its own protected set of data that will be hidden from other end users. This approach allows us to achieve more security. Our approach to implementing this system is to, firstly, plan out a class diagram with the basic objects required. Secondly, to think through the various sequences and ways the program would be used and based on that, to further modify the class diagram. Finally, code the entity, boundary and control classes to create a fully functioning system.

**Principles used:**

Single Responsibility Principle

This principle ensures that each class has only one task and responsibility to execute. For example, the Reservation class manages the attributes and the get set methods. The ReservationController controls the retrieval, creation, and deletion of the Reservation objects. Finally, the ReservationBoundary functions as a User Interface that models the interaction between the user and the system. This concept is applied throughout our application.  
Interface Segregation Principle   
This principle was used to allow a smooth interaction between the user and the system. By separating the user interface into smaller, specific ones, it allows the users to access the methods and options that are of interest to them. This keeps the system decoupled and allow changes to be made easily.

**Assumptions made:**

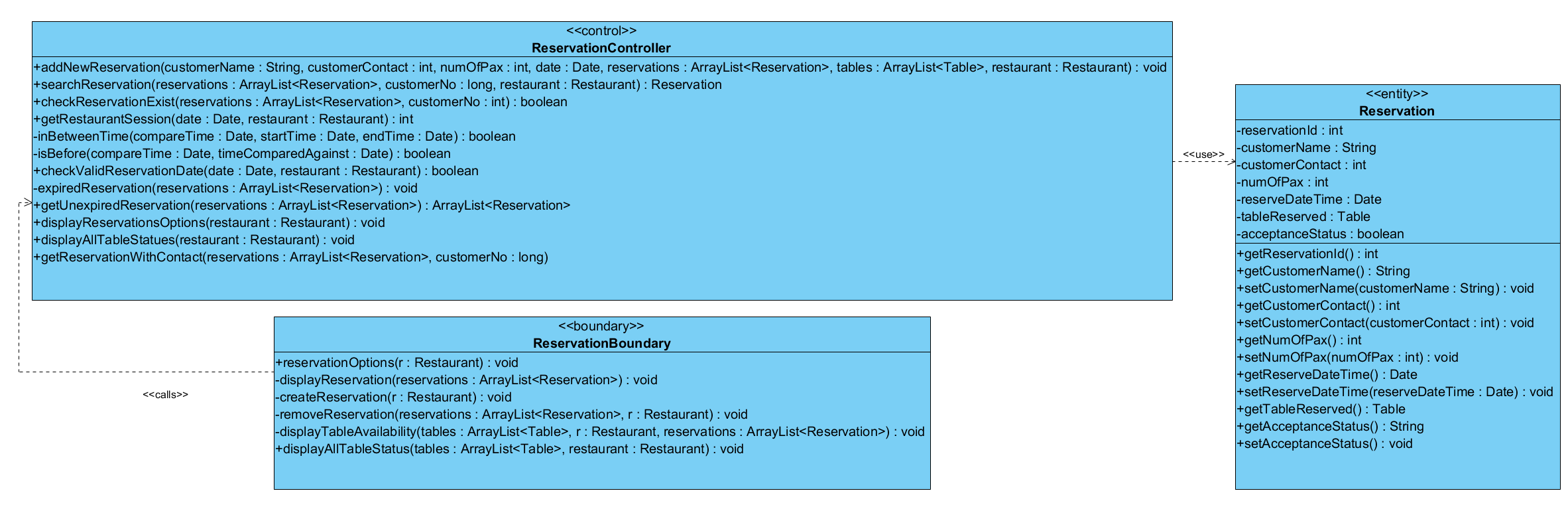
1. Reservation can only be made in at most 1 month in advance.
2. Reservation will be automatically removed if the guest does not arrive within 30 minutes after the stated arrival time\*.
3. The currency will be in Singapore Dollar (SGD) and Good and Services Tax (GST) and service charge must be included in the order invoice.
4. There is no requirement for access control and there is no need for authentication (login/logout) in order to use the application.
5. There is no need to interface with external system, eg Payment, printer, etc. Payment via credit card will always be successful.
6. The restaurant has 30 tables – 5 x 10-seats, 5 x 8-seats, 10 x 4-seats, 10 x 2-seats.
7. Tables cannot be combined/joined (eg, join 3 x 2-seats table to form 6-seats table) and the table ID is fixed.
8. Staff will not use the System (to take orders/reservations etc) outside of operation hours

**Section 2 - Class diagram for the application**

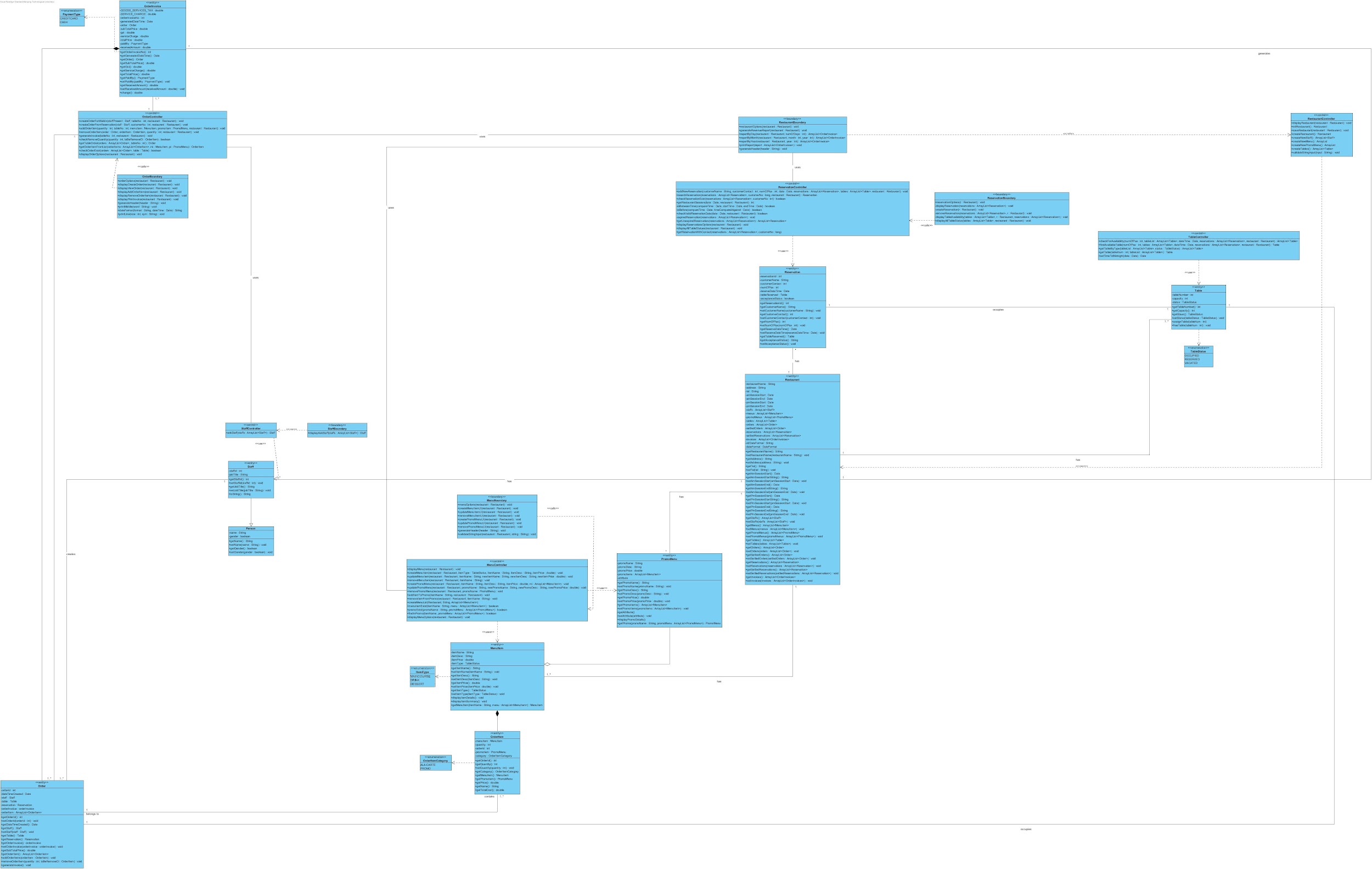
**List of classes:**

| **10 Entity Classes** | | **5 Control Classes** | | **5 Boundary Classes** | | **Enumeration** |
| --- | --- | --- | --- | --- | --- | --- |
| -MenuItem  -Order  -OrderItem  -OrderInvoice  -Person | -PromoMenu  -Reservation  -Restaurant  -Staff  -Table | -MenuController  -OrderController  -ReservationController  -StaffController  -TableController | | -MenuBoundary  -OrderBoundary  -ReservationBoundary  -RestaurantBoundary  -StaffBoundary | | -PaymentType  -OrderItemCategory  -Table Status  -ItemType |

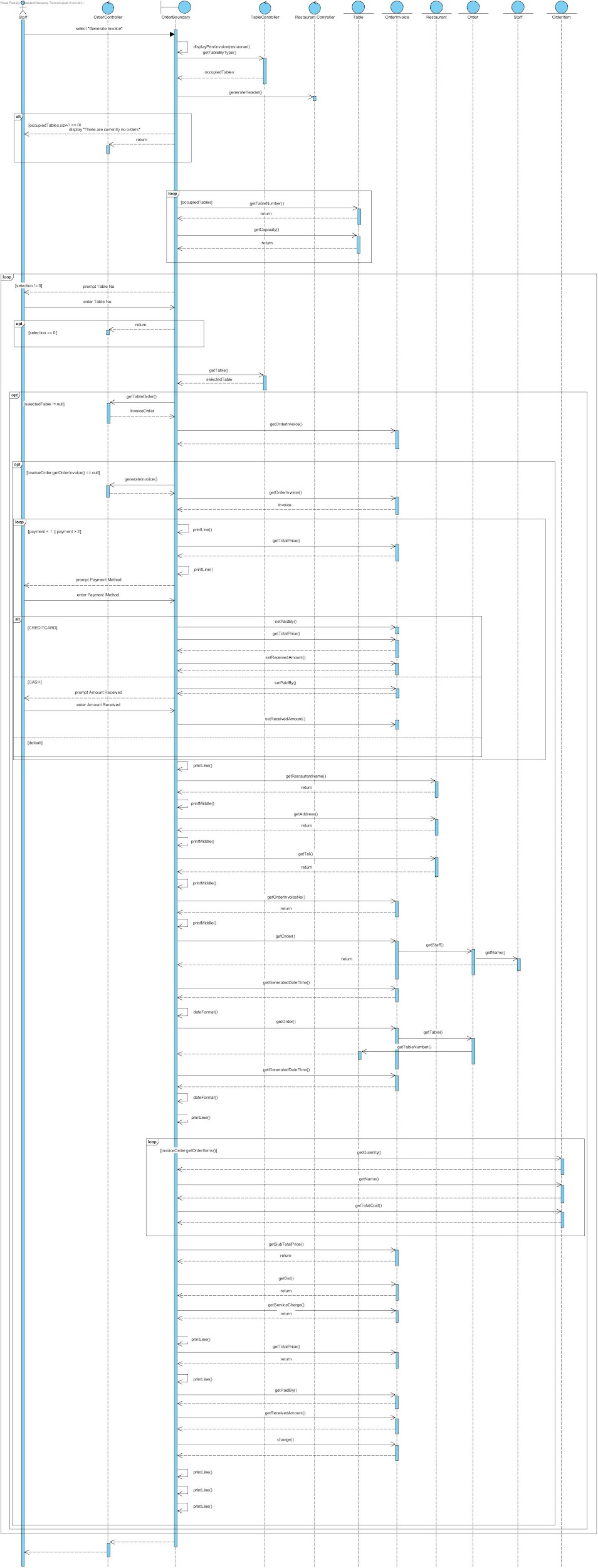
**Example of a Entity-Controller-Boundary relationship:**



**Full Class Diagram (Refer to .jpeg for for quality):**



**Section 3 - UML Sequence diagram for the “Print Bill Invoice” Function**



**Section 4 - Test cases for application**

| **Test** | **Case** |
| --- | --- |
| **Create / Update / Remove item from Menu** | |
| **Objective** | To test the process of creating, updating and removing a menu item |
| **Outcome** |  |
| **Create / Update / Remove item from promotion** | |
| **Objective** | To test the process of creating, updating and removing a promotion |
| **Outcome** |  |
| **Create / View Order** | |
| **Objective** | Create an order and view the created order |
| **Outcome** |  |
| **Add / Remove order item/s to/from order** | |
| **Objective** | To test the adding and removing of an order item |
| **Outcome** |  |
| **Create / View / Delete reservation** | |
| **Objective** | To test the creation and deletion of a reservation and conduct error-checking for any invalid input. Tested for:   * Reserving for a date/time that has past * Reserving during non-operating hours * Reserving for more than a month later |
| **Outcome** |  |
| **Removing of reservation/s upon ‘period expiry’** | |
| **Objective** | To test if the reservation will be automatically removed if the guest does not arrived within 30 minutes after the stated arrival time. |
| **Outcome** | The first reservation checking shows the existence of a reservation booking for 11 Apr 2019, 12:10 PM. (Last on the list)  30 minutes later, as the guest has not arrived, the reservation booking has been automatically removed |
| **Releasing of table/s upon payment / Check table availability** | |
| **Objective** | To test if the status of a particular table will be changed to Vacant after the customer have made their payment by checking table availability |
| **Outcome** | After the guest of Table 5 have made their payment and an order invoice is generated, the table status will be change from occupied to vacant and we are able to see Table 5 in View Available Tables. |
| **Generating bill invoice** | |
| **Objective** | To test if the receipt is correctly generated upon payment. |
| **Outcome** | From View Orders, we can see that these are the items that the customer had ordered.    And when the staff selects Generate invoice, the invoice will be generated |
| **Print sale revenue report by period (eg day or month)** | |
| **Objective** | Test sales revenue report by period |
| **Outcome** |  |

**Section 5 - Consideration for future implementations**

**Additional Feature 1:** Branch Outlet Feature, the System can be used throughout multiple outlets of the same restaurant.

**Specifics:** Upon start up of the application, the staff will be prompted to enter the Restaurant Outlet’s name.

**Implementation:**

This feature can be added into the existing application at ease by simply creating an array of Restaurant classes instead of just having 1 Restaurant Object (current implementation).

**Additional Feature 2:** Membership feature, royalty points given to members which can then be converted to discount when dining at the restaurant.

**Specifics:** 1 royalty point to be given with every dollar spent. Every 100 royalty points can be exchanged for an additional 10% discount to the total bill.

**Implementation:**

Due to the application’s extensibility, we can easily create a Customer class that extends the existing Person class. The newly created Customer class would only require an attribute called “royaltyPoints” with its relevant get and set constructors. When the “displayPrintInvoice” method is being called and the bill is being calculated, the system will check again the customer object to get the available royalty points.