****

**School of Computer Science and Engineering**

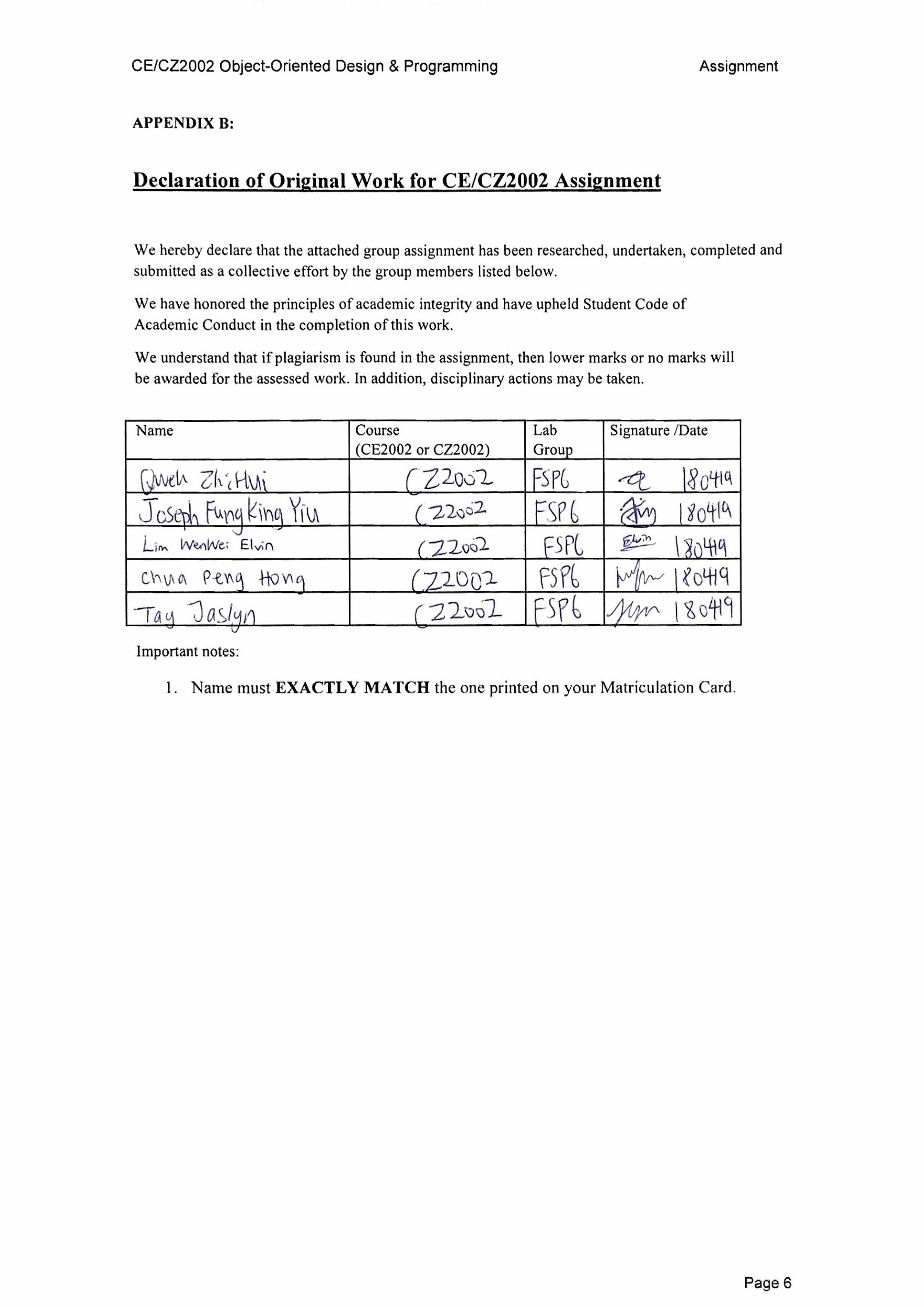
**CZ2002**

**Object Oriented Design Programming**

*Building an OO Application: Restaurant Reservation and Point Of Sale System (RRPSS)*

**Name: Chua Peng Hong(U1721262B), Joseph Fung King Yiu(U1822415L), Lim WenWei, Elvin(U1822943H), Qwek Zhi Hui(U1822304C), Tay Jaslyn(U1721893B)**

**Group: FSP6 Group 1**



## 

## Design Considerations

The very first step we take is identifying important classes for the system and group them into Entity Class, Control Class and Boundary Class.

**Assumptions** (On top of those stated in Assignment Brief):

* 1 customer can only make 1 reservation for the same session and on the same day, but he can have multiple reservations. E.g He can book morning session and afternoon session
* Allocation of tables based on best fit and first come first served. 3 pax will be allocated to 4-seats table, and if there are no more 4-seats table, allocated to 6-seats table, and so on.
* The program runs 365 days a year with no breaks in between from 11am - 3pm (AM Session) and 6pm-10pm (PM Session).
* The staff will always print the bill invoice for every table, to mark the sale as complete.
* Calling of Remove Reservation will remove all Reservations linked to a particular contact number.

We used a 4-Layered Architecture for our system to hide our implementation of methods from users and to also prevent them from directly interacting with the data files.

1. [Boundary] Presentation Layer (main application interface)

2. [Control] Business Layer (control classes i.e. MenuMgr, ReservationMgr, OrderMgr etc.)

3. [Entity] Persistent Layer (entity classes i.e. Menu, Reservation, Order etc.)

4. Database Layer (data text files i.e. MenuItems.txt, Reservations.txt, Orders.txt)

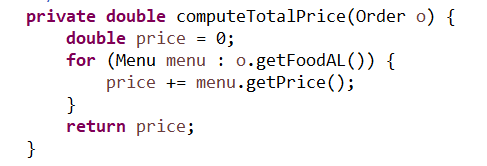
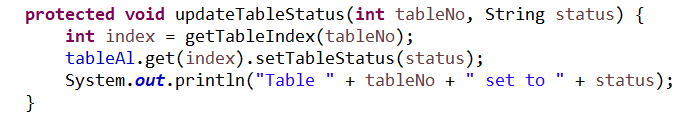
We have decided to use this architecture because

1. It increases flexibility and reusability. Any changes that are made to a class in a specific layer would not affect the other classes from other layers. For example, if we were to change our interface from a CLI (Command Line Interface) to GUI (Graphical User Interface), we just need to update the main application interface class in the presentation layer. All other classes from other layers remain the same.

2. It also encourages extensibility and maintainability. The layers in the architecture are independent, they can be individually updated, deployed and maintained.

3. It makes the system loosely coupled since the layers must communicate via top-down approach.

To further bolster the 4-layered system architecture, Access Level Modifiers are used. Access level modifiers determine whether other classes can use a particular field or invoke a particular method. Methods in our project that prevents access from other classes have a protected modifier. The protected modifier specifies that the member can only be accessed within its own package “Control” and, in addition, by a subclass of its class in another package,



Within each class, a private modifier is used to restrict access to itself. The private modifier specifies that the member can only be accessed in its own class “InvoiceMgr”.

### Use of OOP Design Principles

1. Single Responsibility Principle

We make sure that every single class has only a single responsibility. For example, the ReservationMgr class has methods relating with reservations only. Methods within ReservationMgr handles reservations exclusively and not other areas of the system.

1. Open-Closed Principle

We implement this principle for the Menu class. The Menu class has a “Category” interface. To add more food categories into the menu, we do not need to edit code of the class, we just add a new food category class realizing “category” interface.

1. Interface Segregation Principle

The 4-layered architecture allows proper segregation of classes. These classes are grouped together and connected to a specific interface. All of the classes are connected to interfaces that they rely on, each interface serving a specific purpose.

Consideration for Further Enhancements

1. Block-booking of entire session for private events [flexibility]

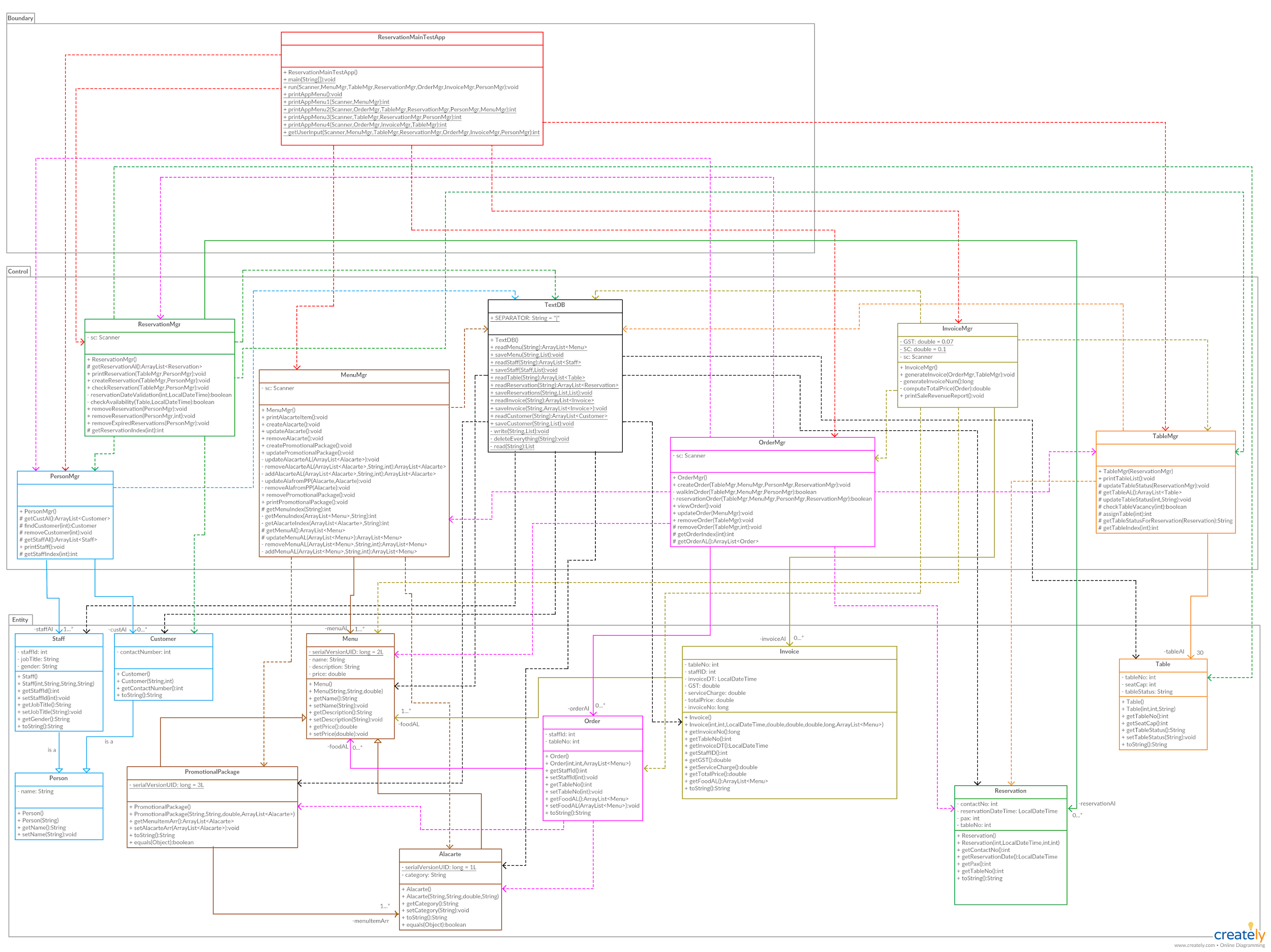
By implementing a system whereby block-booking for events can take place, it will become easier to facilitate high capacity reservations. Our existing system is flexible as the addition of another method to reserve the whole venue would not impact the functionality of other classes in any layers. The 4-layered architecture results in loose coupling between entity and control classes as it promotes interaction via interfaces.

1. Waiting list for tables [extensibility]

A waiting list system for diners looking to book a reservation during peak hours/when the restaurant is fully booked. If tables are fully booked, customer may opt to be put on a waiting list. Implementation of this system would be easy as the 4-layered architecture improves extensibility. Creation of an additional class within a layer is an independent action and would not impact other classes in other layers.

# 

# Detailed UML Class Diagram



Due to the large image resolution, please view the image inside the folder (Class Diagram.jpg)

# Detailed UML Sequence Diagram of “Print invoice”

# Test cases and results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | Test Cases | Input | Expected Results | Actual Results | Test Case Result (Pass/Fail)  /Screenshots |
| 1. | View menu | View menu | Success | Success |  |
| Create/Update/Remove menu item | | | | | |
| 2a. | Creating menu item | Add fried rice at $3 | Creation of menu object is successful | Creation of menu object is successful |  |
| 2b. | Updating non-existential menu item | Update fried noodles to $4 | Display Error Message | Display Error Message |  |
| 2c. | Updating existing menu item | Update fried rice to $5 | Success | Success |  |
| 2d.. | Removing existing menu item | Remove fried rice | Success | Success |  |
| Create/Update/Remove promotions | | | | | |
| 3a. | Creating new promotion | Create lunch promotion | Success | Success |  |
| 3b. | Updating non-existing promotion | Update dinner promotion | Display Error Message, brought back to main page | Display Error Message, brought back to main page |  |
| 3c. | Updating existing promotion | Add 2 drinks to lunch promo | Success | Success |  |
| 3d. | Removing non-existential promotion | Remove dinner promo | Display Error Message, brought back to main | Display Error Message, brought back to main |  |
| 3e. | Removing existing promotion | Remove lunch promo | Success | Success |  |
| Create/Update/Remove Order | | | | | |
| 4ai. | Walk-in | Create walk-in order of Carbonara Pasta and Milkshake. | Success | Success |  |
| 4aii. | Walk-in (fully-booked) | Walk-in order when tables are occupied | Display Error Message, Prompt to re-enter | Display Error Message, Prompt to re-enter |  |
| 4bi. | Reservation | Order based on reservation (contact’s phone number) | Success | Success |  |
| 4bii | Reservation (invalid phone number) | Enters wrong reservation number | Display Error Message, Prompt to re-enter | Display Error Message, Prompt to re-enter |  |
| 4c | Invalid staff ID | Enters wrong staff ID | Display Error message, goes back to mains screen. | -Display Error message, goes back to mains screen. |  |
| 4d. | Update vacant table |  | Display Error Message, Prompt to re-enter | Display Error Message, Prompt to re-enter |  |
| 4ei. | Add items to order (occupied) | Add banana soup to table 1 | Success | Success |  |
| 4fii. | Remove items from order (occupied) | Remove mushroom soup | Success | Success |  |
| 4g | Remove order from vacant table | Remove table 10, vacant table | Display Error Message, Prompt to re-enter | Display Error Message, Prompt to re-enter |  |
| View Order | | | | | |
| 5a. | View vacant table | Views vacant table | Display Error Message, Prompt to re-enter | Display Error Message, Prompt to re-enter |  |
| 5b. | View occupied table | Views table 1, with mushroom soup order | Success | Success |  |
| Create/Check/Remove reservation booking | | | | | |
| 6a. | Create reservation booking when fully-booked | Create when fully booked | Display Error Message | Display Error Message |  |
| 6b. | Create reservation booking with wrong time (ahead) | Input date as 10-07-19 | Display Error Message | Display Error Message |  |
| 6c. | Create reservation booking successfully | Create reservation with correct time | Success | Success |  |
| 6d. | Check existing booking | Checks using reserved phone no. | Success | Success |  |
| 6e. | Remove non-existent booking | Use wrong phone number to remove | Display Error Message, Prompt to re-enter | Display Error Message, Prompt to re-enter |  |
| 6f.. | Remove existing booking | Use correct number to remove | Removes all reservations under the phone number. | Removes all reservations under the phone number |  |
| View Tables | | | | | |
| 7a. | Table Availability | Shows table availability | Success | Success |  |
| Print bill invoice | | | | | |
| 8a. | Print invalid table invoice | Print wrong table | Display Error Message, Prompt to re-enter | Display Error Message, Prompt to re-enter |  |
| 8b. | Print valid table invoice | Print correct table | Success | Success |  |
| Print sale revenue report by period(e.g day/month) | | | | | |
| 9a.. | Print day report | Print today’s report | Success | Success |  |
| 9b. | Print month report | Print monthly report | Success | Success |  |