**Demonstration video link :**

[**https://www.youtube.com/watch?v=IxGTpbQ865E**](https://www.youtube.com/watch?v=IxGTpbQ865E)

**Section 1:** **Design considerations and use of OO concepts.**

**1.1 Approach Taken**

In the initial stages of designing the RRPSS application, we had decided that it would be best if the application was designed to be as practical and realistic as possible. This was an important approach as it helped to shape the structure of our application and also served as a guideline on how we think the user should interact with the application. However, as compared to a real-life application with similar functionalities, our version of the application does not come with a pre-existing database that we can build upon. As such, functions like addMenuItems() were necessary in order for our code to work.

Based on the functional requirements that were given to us, we took careful consideration on what are the Classes that are needed to meet those requirements. Most importantly, we had to make sure that Classes relating to the tables, reservations and orders in the restaurant are defined clearly as these are the foundation of our application.

**1.2 Assumptions Made**

There are a number of assumptions that we have to make in order to simplify the problem and at the same time be able to simulate real world use. They are as follows:

* Restaurant is by reservation only. No walk-ins.
* Only same day reservations are allowed
  + This allows for better management of tables
* Reservations expire 10 minutes after reservation timing.
* Order will be paid after printing order invoice ( selection 10 in the code)
* Orders are applicable for GST and service charge.
  + In our case, we combined the two components and made it into a 17% charge on the total price.
* Promotion sets must contain one item from each category. (Main Dish, Desserts, Beverages).

**1.3 Design Principles Used**

As taught in Design Principles, a good design is one that aims to achieve loose coupling and high cohesion between the different classes while still being able to achieve a design that has high reusability , extensibility and maintainability. Another good principle would be good encapsulation where each class is solely responsible for its function and does not require the knowledge of how a separate class operates.

While designing the application, we did our best to align our design as closely to the SOLID design principles as possible.

**Single Responsibility Principle**

This principle states that a single class should only have one responsibility. This reduces the need to change more than one class when change is applied to one class and hence reduces the snowball effect. An application of this principle can be seen in our prePopulate Class where its only responsibility is to provide the application with a dataset to work on.

**Open-Closed Principle**

The main idea of this principle is that Abstraction is the key to OODP and that “a module should be open for extension but closed for modification”. This allows certain classes to extend more while keeping the methods consistent.

In our project, we have an OrderList class which is the parent class of SalesRecord. The OrderList class contains an arraylist of Order objects. With this OrderList class, if we were to add on new classes that require an arraylist of Order objects and one or more methods in the OrderList class, we can make them subclasses of OrderList. This way, OrderList is closed for modification, but open for extension for new classes.

**Liskov Substitution Principle**

The Liskov Substitution Principle states that if a derivative of a base class is passed to a user of the base class, the user should continue to function properly. This essentially means that the derived class can be used as a substitute for its base class and when passed to a subclass of the base class, it will still be able to meet the method’s pre and post conditions.

**Interface Segregation Principles**

A supposedly general interface where many classes implement its methods should not need to implement methods that it does not use. This is the main idea of this principle. As such, interfaces should be separated based on their specific functions.

**Dependency Injection Principle**

This principle states that high level modules should not depend upon low level modules but instead both modules should depend upon abstractions. This provides support for pluggable implementations hence allows for the building of loosely coupled components.

The applications of these principles in our design are usually interleaving and are designed concurrently. As such, it can be seen that some of our code designs have a few of these principles applied simultaneously.

**1.4 Use of Object-Oriented Concepts**

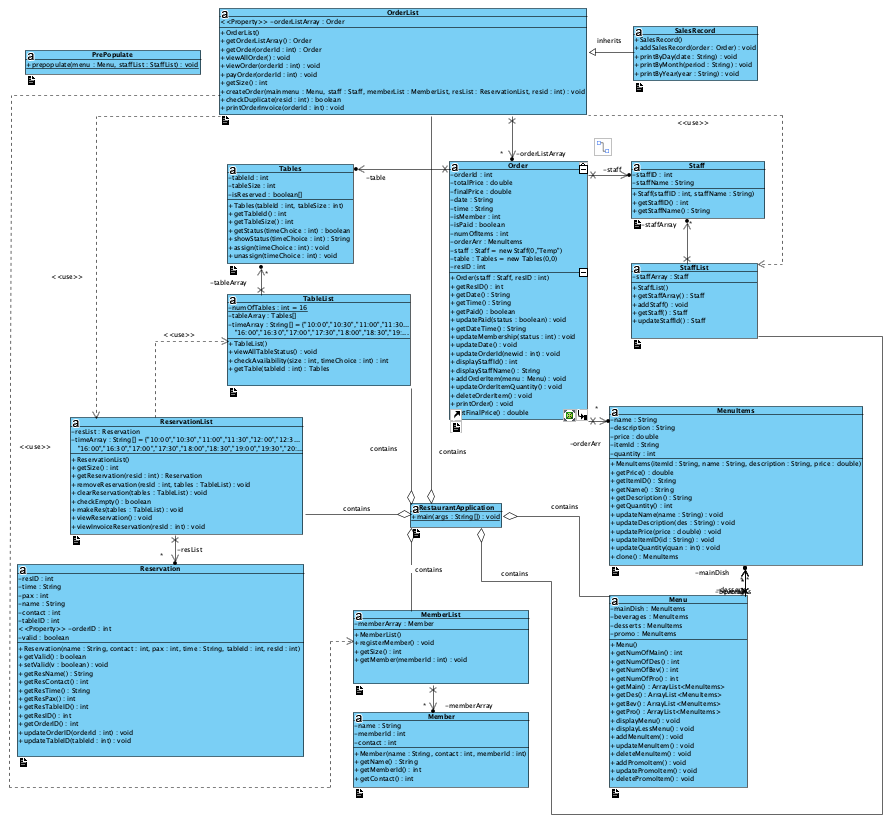
**Abstraction** and **Encapsulation** was used to ensure that the data attribute within a class remained private to ensure that it could not be changed, and public methods were used within the class to gain access to them. This way, we are providing only essential information outside of the class while hiding their background details or less essential information.

**Inheritance**

Used for the SalesRecord Class extending the OrderList Class as SalesRecord is viewed as a compilation of all the orders that are received and already paid by the customer.

**Section 2: UML Class Diagram**

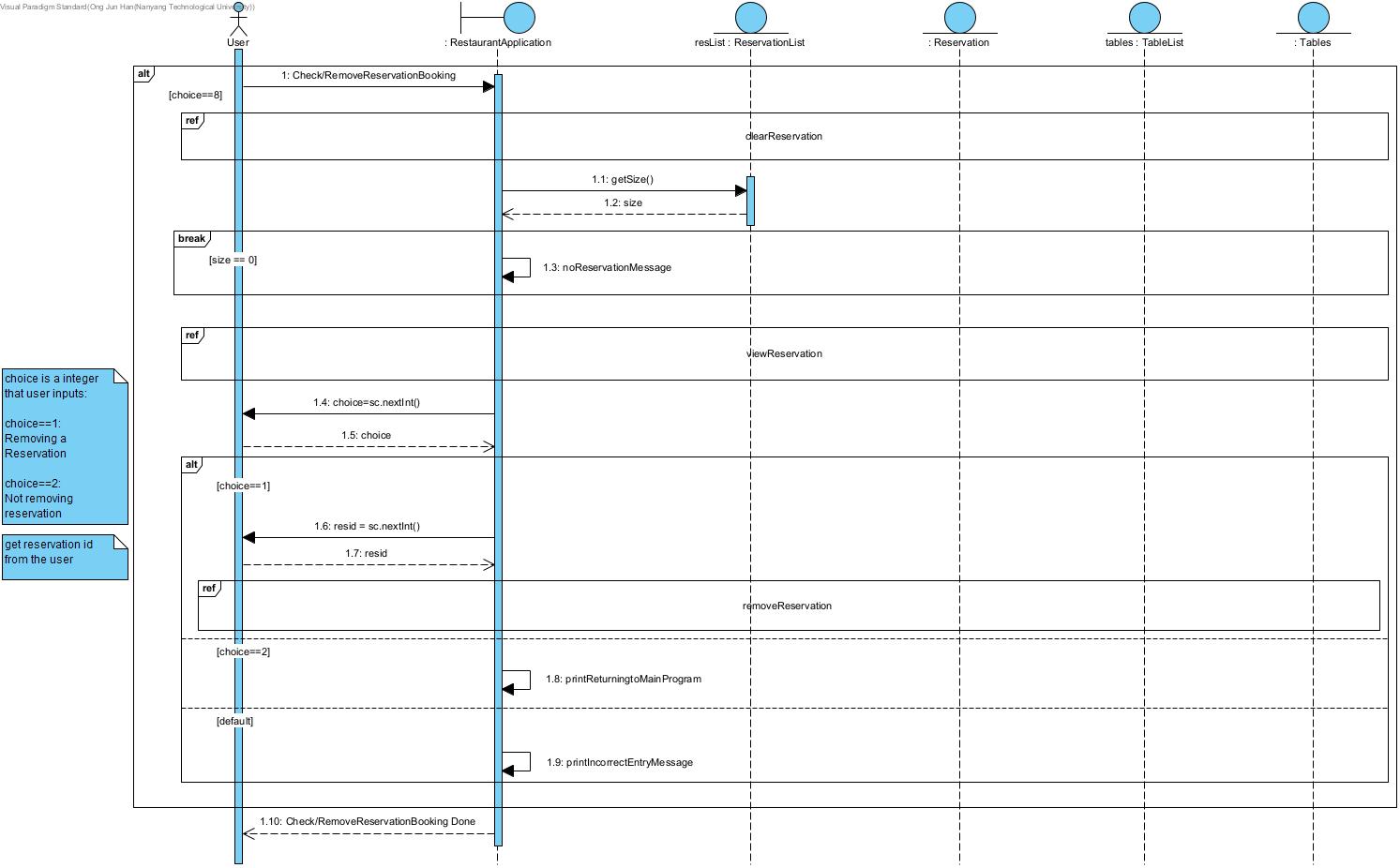
**2.1 Whole Diagram**

****

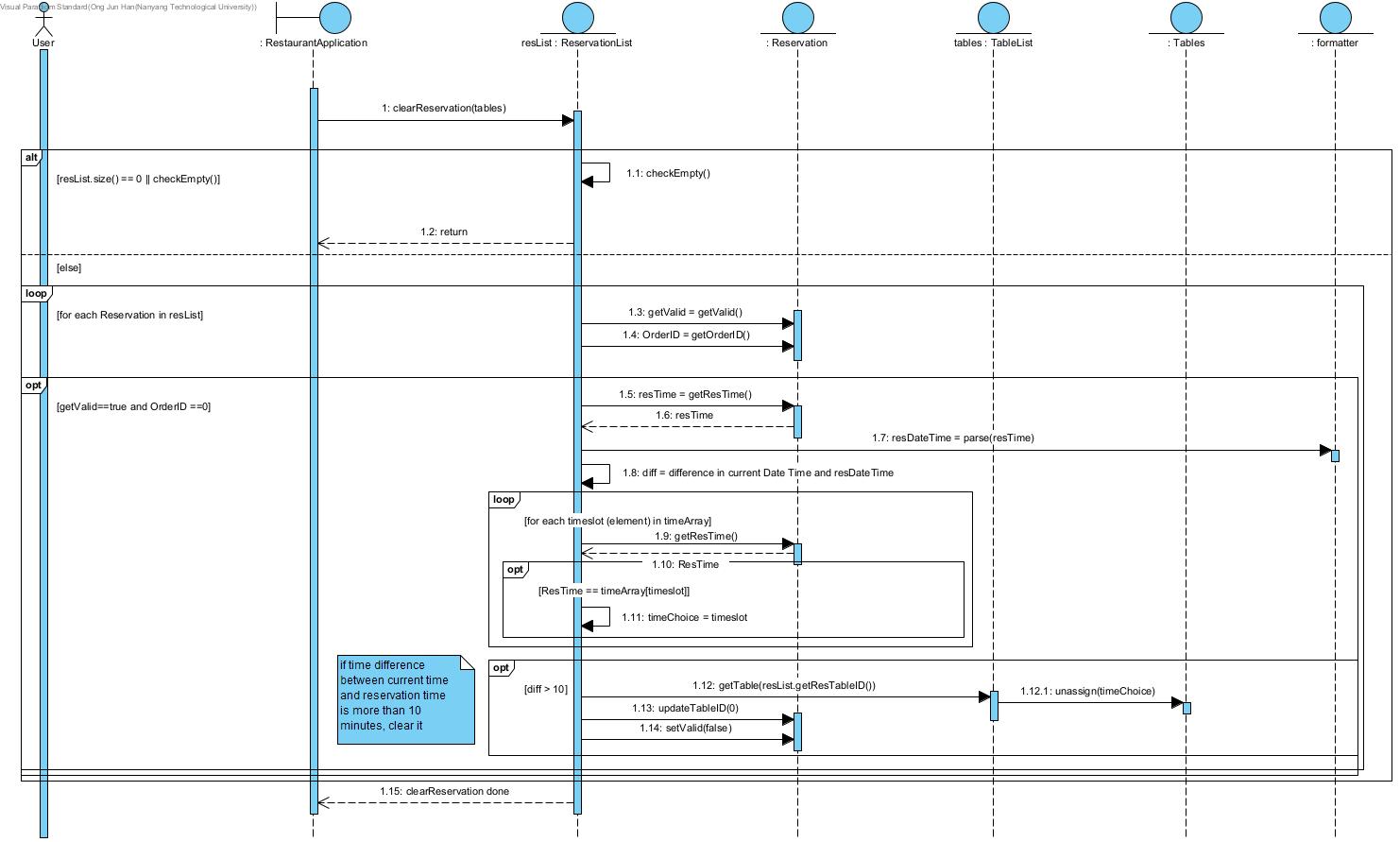
**Section 3: UML Sequence Diagram**

**3.1 Diagram:**

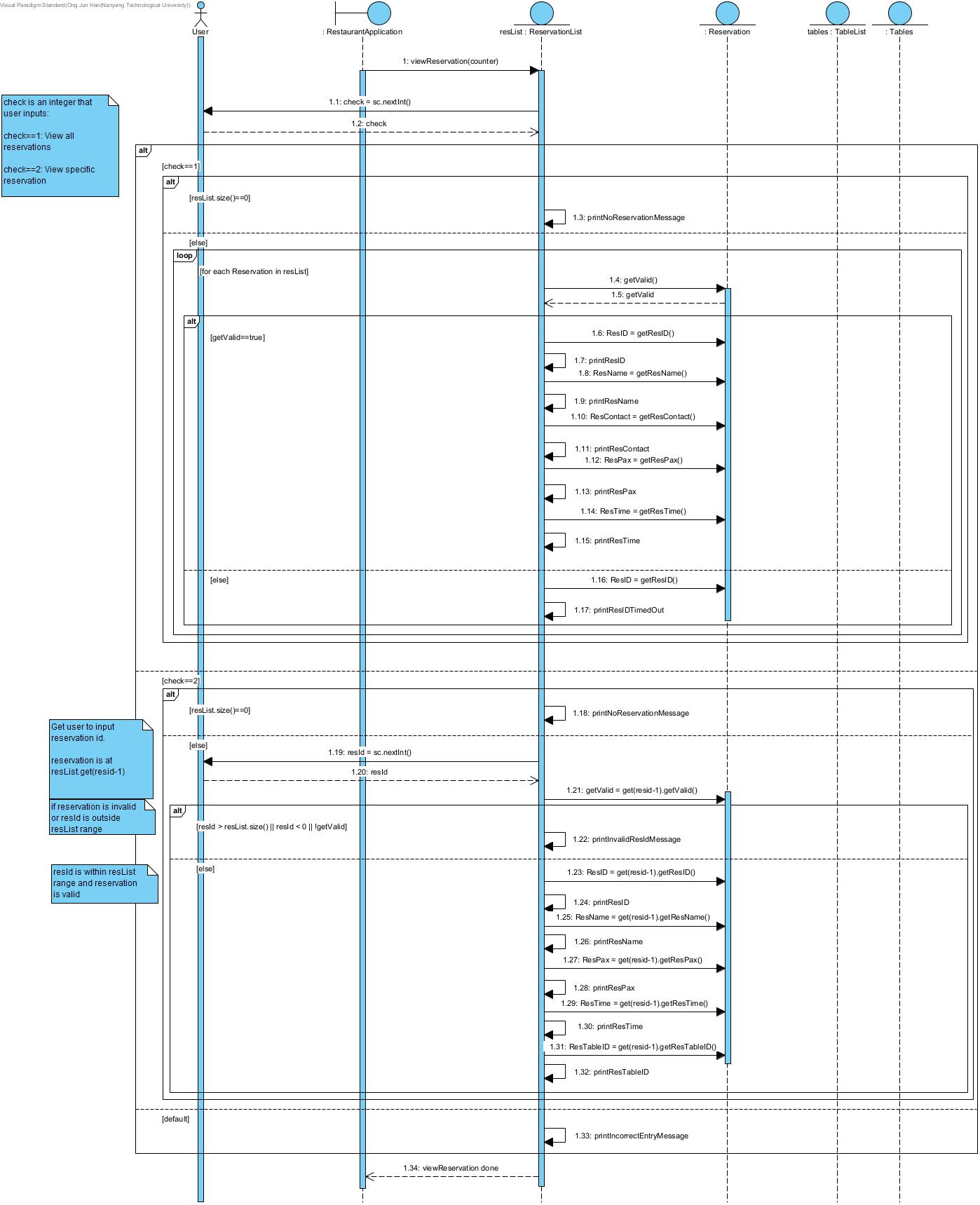
1. **Check/Remove Reservation main diagram**

****

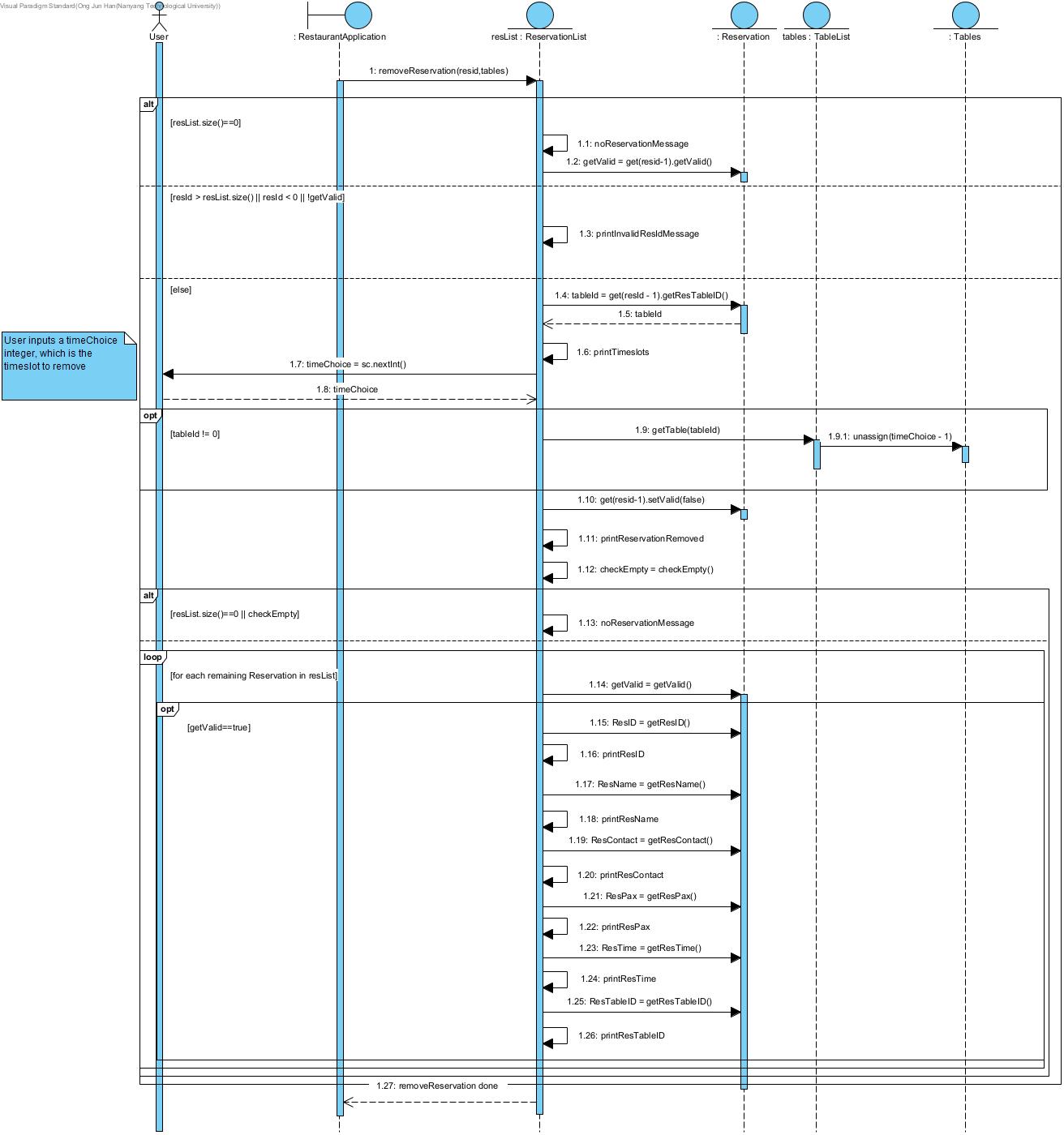
1. **clearReservation reference frame**

****

1. **viewReservation reference frame**

****

1. **removeReservation reference frame**

****

**3.2 Explanation**

Within the main check/remove reservation booking function, there are 3 major components: clearReservation, viewReservation and removeReservation.

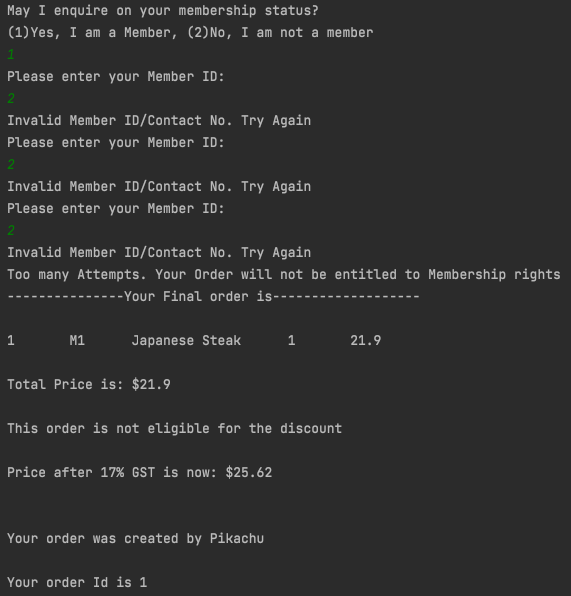
The user selects the choice 8, which is the check/remove reservation booking function, where the boundary object or application RestaurantApplication will carry out the following:

clearReservation is done first, removing the reservations that have reached ‘period expiry’, then viewReservation is done next to print out all current reservations and their corresponding details, followed by removeReservation, which allows the user to manually remove any current reservation.

In the sequence diagrams, exception handling is not shown and included for more concise diagrams. User inputs are also shown and the choices for these inputs are explained within the sequence diagrams. All print self-messages are simple print outs of the values they are assigned to print.

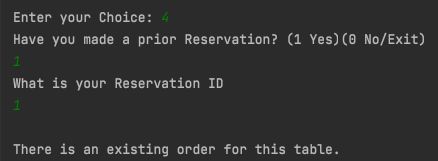
**Section 4: Testing Done**

**Test 1: Entering non existing Member ID**

****

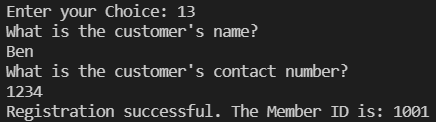
Description: “2” is not a valid MemberID, after three times of attempt, it will be rejected and the order will be treated as a non-member order.

**Test 2: Making second order for a Reservation ID**

****

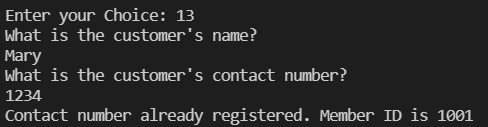
Description: We have made an order for Reservation ID “1” , hence we cannot make another one.

**Test 3: Successful registration of member**

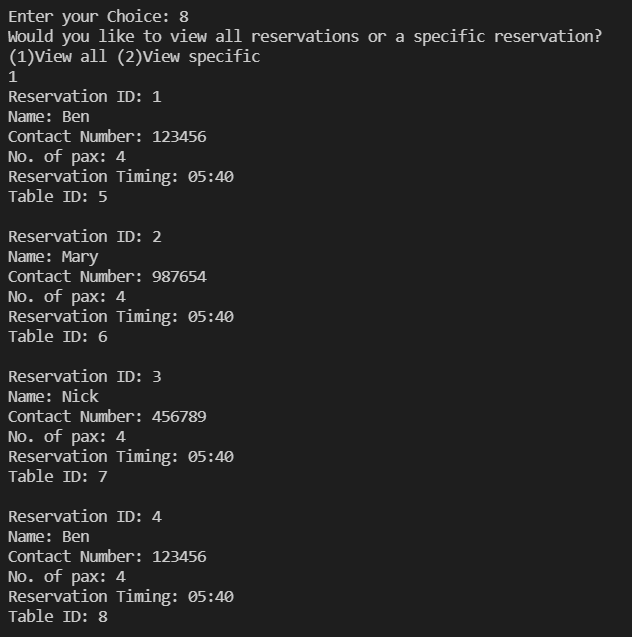
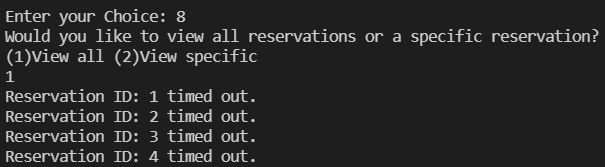
****

Description: Show successful member registration.

**Test 4: Register existing member**

****

Description: Continue from Test 3. Allow contact numbers to be only registered once. Therefore, a duplicate contact number will return invalid registration.

**Test 5: Clear Reservation function**

Description: Screenshot on the left (taken before 5.50) shows existing reservations. Screenshot above (taken at 5.51) shows that reservations have timed out and are invalid.

Note: Time options (for makeReservations) were changed to facilitate testing as clearReservation() takes current time. Results are 100% genuine.

**Cover Page:**

**Declaration of Original Work for CE/CZ2002 Assignment**

We hereby declare that the attached group assignment has been researched, undertaken, completed and submitted as a collective effort by the group members listed below.

We have honored the principles of academic integrity and have upheld Student Code of Academic Conduct in the completion of this work.

We understand that if plagiarism is found in the assignment, then lower marks or no marks will be awarded for the assessed work. In addition, disciplinary actions may be taken.

| Name | Course (CE2002 or CZ2002) | Lab Group | Signature/Date |
| --- | --- | --- | --- |
| Muhammad Irfan Bin Norizzam | CZ2002 | SS1 | 14/11/2021 |
| Ong Jun Han | CZ2002 | SS1 | 14/11/2021 |
| Chai Wen Xuan | CZ2002 | SS1 | 14/11/2021 |
| Yong Chi Kei | CZ2002 | SS1 | 14/11/2021 |