

**APPENDIX B:**






Attached a scanned copy with the report with the filled details and signatures.

**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
Yang Yang	CZ2002	SS6	 13/11/2021
Hans Farrell Soegeng	CZ2002	SS6	 13/11/2021
Fion Chai Xin Yi	CZ2002	SS6	 13/11/2021
Foo Wei Ling	CZ2002	SS6	 13/11/2021
Joey Quah (Ke Jieyi)	CZ2002	SS6	 13/11/2021

Important notes:

1. Name must **EXACTLY MATCH** the one printed on your Matriculation Card.

## **1. DESIGN CONSIDERATIONS**

### **1.1 Approach Taken**

Our Restaurant Reservation and Point of Sale System (RRPSS) Application's main goal is to digitalize all the essential operations in a restaurant, so as to smoothen out the chain of operations, as well as to increase the efficiency of work. Our group designed this application with restaurant staff as our target user, hoping to bring them an application which is user friendly, easy to navigate and practical at the same time.

After analysing the question, we established 6 fundamental classes - Table, Order, Menu, Reservation, Employee and Sales Revenue. These classes make up the basis of our application and enable the construction of the essential functionalities of the system that is required for real world operation.

From there, we introduced other concrete and abstract classes to support these 6 classes, such that we can satisfy the Design Principles, which we will be discussing in the next section.

(Here is the link to our video demonstration: <https://youtu.be/XmafbeiUd3M>)

### **1.2 Principles used**

When designing the application, we made use of the SOLID Design Principles as well as some Object-Oriented (OO) Concepts.

#### **1. Single Responsibility Principle**

The application is built such that each class is only responsible for one main function, preventing the application from having classes with coupled responsibilities. This prevents the need for making changes to a class for more than one reason. The OO Concept of Encapsulation is also applied here as the main responsibility of each class is enclosed entirely within the private methods of the class.

As mentioned earlier, we introduced more supporting classes in addition to the six main classes, such that each class will have only one primary responsibility. A detailed illustration of how the Single Responsibility Principle is applied to each main class is shown below.

Main Class	Supporting Classes	Responsibility
Reservation	Reservation	Create reservations
	ReservationPrinter	Printing the reservation details
	ReservationChecker	Checks the details of reservations to determine whether they have expired
Table	Table	Stores information of individual tables
	TablePlanner	Manages all tables in the restaurant
Order	Order	Stores the orders for each table
	OrderInvoice	Prints the order invoice when a customer is making payment
Menu	Menu	Prints the main menu
	MenuItems	Stores information of individual menu items
	MenuUpdater	Edits the menu
Employee	Employee	Stores each employee's information
	EmployeeRoster	Stores consolidated list of all employees
SalesRevenue	SalesRevenue	Consolidates all the orders of the day
	SalesRevenueRecorder	Stores and prints sales revenue reports

## 2. Open-Closed Principle

The Open-Closed Principle states that “a module should be open for extension but closed for modification”. This allows the attributes and operations of a class to be extended without modifying the source code. This is satisfied by the use of abstract classes in our application.

The OO Concept of Abstraction is also applied here. Abstraction is the process of showing only essential information. The abstract methods in the abstract class are only implemented in the subclasses, hence only showing essential details in the abstract class and hiding the implementation details.

One example would be the MenuItem Class, an abstract class that can be extended by subclasses such as the MainCourse, Dessert, Drinks and PromotionSet classes. With such an implementation, whenever the restaurant wants to add another category of menu items, users are able to do so by simply extending a new class from the MenuItem class, without having to modify the MenuItem class.

Similarly, the staff members of the restaurant can also introduce new methods and attributes to a single category by modifying its subclass. For example, the Drinks class can have a temperature attribute, like hot or cold drinks, which is not shared by the other subclasses. The implementation details of these new methods and attributes will not be shown in the abstract superclass.

### 3. Liskov Substitution Principle

The Liskov Substitution Principle says that objects should be easily replaceable with instances of its subtypes to perform the same methods, without altering the correctness of the program.

In our application, we made use of abstract classes, such as the Table class, which stores information of each individual table. It is extended by the TwoSeats, FourSeats, SixSeats, EightSeats and TenSeats subclasses, which differ by the number of seats available at each table. These subclasses are able to substitute the Table class as the methods initialised by the superclass are inherited by the subclasses. As such, when creating a new Table object, we can instead instantiate its corresponding subclass, as shown in the code below.

```
public TablePlanner() {
    //Table[] tables = new Table[21];
    for(int i = 0; i < 10; i++) {
        tables[i] = new TwoSeats(i);
    }
    for(int i = 10; i < 16; i++) {
        tables[i] = new FourSeats(i);
    }
    for(int i = 16; i < 18; i++) {
        tables[i] = new SixSeats(i);
    }
    for(int i = 18; i < 20; i++) {
        tables[i] = new EightSeats(i);
    }
    tables[20] = new TenSeats(20); }
```

The OO Concept of Inheritance is also applied here because the Table class and its subclasses form a hierarchy of classes that share a set of attributes and methods.

#### **4. Interface Segregation Principle**

Specific interfaces should be used to ensure that classes do not depend on general interfaces with methods they do not use. The Interface Segregation Principle does not apply to our application as it did not make use of any interfaces. This is because every main class has distinct functions and is supported by other classes. Instead of interfaces, we chose to use abstract classes since the methods of the subclasses are identical, therefore can be inherited from the superclass rather than implementing them separately.

#### **5. Dependency Injection Principle**

Abstract classes are used in the building of our application to ensure the loose coupling of software modules and reduce the coupling of components. As such, high level modules are independent of low level modules and high level modules can be reused easily without the need to edit much. For example, abstract class MenuItem can be depended on by other classes, like the Menu class.

### **1.3 Assumptions Made**

Below are the assumptions made for our application, besides those specified in the assignment.

- Menu will not be edited during opening hours.
- Editing orders can only be done such that the customer's final order will have at least one item.
- Reservation can only be made at most 4 hours in advance.
- Once a reservation is made, the table cannot be seated unless the reservation is removed or the customer has arrived.
- Sales need to be closed before sales revenue is calculated.
- Only close sales at the end of the day and after all tables are empty. Any orders added after closing sales will be assumed as the next day's sales.

## 2. DETAILED UML CLASS DIAGRAM

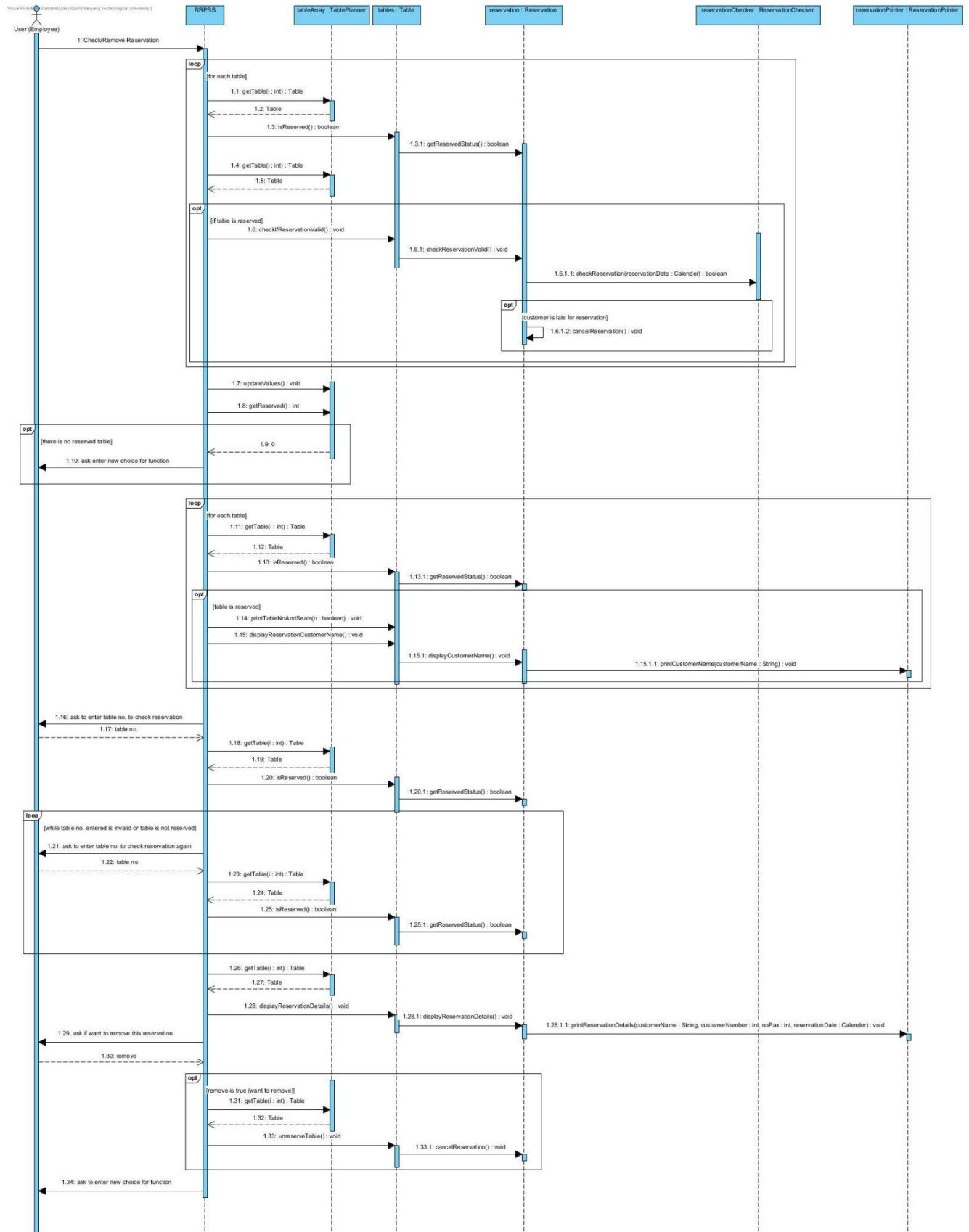


- A composition relationship is an ‘is part of’ association relationship. This relationship is formed when one class cannot exist without the other. For example, the MenuItem class has a composition relationship with the Menu class as each menu item is part of the menu, and only exists if there is a menu. Similarly, the Order class has a composition relationship with the Table class because every table has an order, and the order would not exist without a table.
- An aggregation relationship is also an ‘is part of’ relationship, but is formed when both classes can exist even if the other class is destroyed. For example, the SalesRevenue class has an aggregation relationship with the SalesRevenueRecorder class as the sales revenue reports generated are contained in the sales revenue recorder. Even if the sales revenue or sales revenue recorder is removed, the other class can still exist on its own.
- A unidirectional association relationship arises when only one class needs to acquire information from the other class. For example, the Menu class and Order class have a unidirectional association relationship, from the Order class to the Menu class. This is because the order obtains information from the menu, but the menu does not need any information from the orders.

### **3. DETAILED UML SEQUENCE DIAGRAM (Check/Remove Reservation)**

The following is the flow of sequences when a user wants to check or remove reservation.

1. When a user selects the “Check/Remove Reservation” function, the application will first remove all the invalid reservations (i.e. the customer is late for his reservation) and check if there are any reserved tables.
2. Next, the application will show all the reserved tables with the corresponding customer names and ask which reservation the user wants to check. The application will then print the details of the reservation.
3. Lastly, the application will ask if the user would like to delete the reservation. If yes, the reservation will be deleted.





## **4. FURTHER TEST CASES**

The following are the test cases that we did not include in our video demonstration.

### **4.1 Add/Remove Employee**

Our application has a function that allows users to add and remove employees from the restaurant's employee roster. Figure 1 shows an example of adding a new employee to the roster, while Figure 2 shows an example of removing an employee from the roster.

```
11
===== Employee Menu =====
|1. Add new Employee          |
|2. Remove Existing Employee  |
|3. Print Employee particulars|
|4. Done and Exit            |
=====

Please select one option to proceed
1
Enter new Employee name:
William
Enter new Employee gender:
M
Enter new Employee Job Title:
Waiter 7
Enter new EmployeeID:
1009
Done!
```

Figure 1: Adding a new employee

```
11
===== Employee Menu =====
|1. Add new Employee          |
|2. Remove Existing Employee  |
|3. Print Employee particulars|
|4. Done and Exit            |
=====

Please select one option to proceed
2
Enter EmployeeID to be removed:
1003
Done!
```

Figure 2: Removing an employee

### **4.2 Exception Handling**

In our application, we included several exception handlers to handle all the unintentional invalid inputs by the users. The following test cases show the error messages that will be displayed when there are invalid inputs from users.

#### **4.2.1 No existing orders to edit**

Case for when a new day begins and no orders have been made yet, but the user tries to edit, view or print an order. An error message will be shown, in Figure 3, to indicate that there are no orders yet.

```

===== Please choose one =====
| 1. Edit Menu Item |
| 2. Create Order   |
| 3. Edit Order     |
| 4. View Order     |
| 5. Create Reservation Booking |
| 6. Check/Remove Reservation Booking |
| 7. Check Table Availability |
| 8. Print Order Invoice |
| 9. Close Sales    |
| 10. Print Sales Revenue Report By Period(by day) |
| 11. Add/Remove Employee |
| 12. Exit          |
=====
3
There are currently no orders.
4
There are currently no orders.
8
There are currently no orders.

```

Figure 3: Error message: “There are currently no orders.”

## 4.2.2 Fully Seated

Case for when the restaurant is fully seated and the user either tries to create a new order (Figure 4), or create a new reservation (Figure 5). An error message will be shown for each of the cases respectively.

```

===== Please choose one =====
| 1. Edit Menu Item |
| 2. Create Order   |
| 3. Edit Order     |
| 4. View Order     |
| 5. Create Reservation Booking |
| 6. Check/Remove Reservation Booking |
| 7. Check Table Availability |
| 8. Print Order Invoice |
| 9. Close Sales    |
| 10. Print Sales Revenue Report By Period(by day) |
| 11. Add/Remove Employee |
| 12. Exit          |
=====
2
Restaurant is fully seated! You can no longer create orders

```

Figure 4: Error message: “Restaurant is fully seated! You can no longer create orders.”

```

===== Please choose one =====
| 1. Edit Menu Item |
| 2. Create Order   |
| 3. Edit Order     |
| 4. View Order     |
| 5. Create Reservation Booking |
| 6. Check/Remove Reservation Booking |
| 7. Check Table Availability |
| 8. Print Order Invoice |
| 9. Close Sales    |
| 10. Print Sales Revenue Report By Period(by day) |
| 11. Add/Remove Employee |
| 12. Exit          |
=====
5
Restaurant is full! You can no longer create reservation bookings.

```

Figure 5: Error message: “Restaurant is full! You can no longer create reservation bookings.”

## 4.2.3 Out of Menu Bounds

Case for when the user tries to add a menu item that does not belong to the menu to their order. In Figure 6, the user tries to add an order from category index 4, which does not exist in the menu, thus receiving an error message as shown.

```

What category?
4
Which item?
5
How many?
2
Index 4 out of bounds for length 4

Plases choose again!

```

Figure 6: Error message: IndexOutOfBound error message

Case where the user tries to remove an item that is not in the menu. An error message will be shown as shown in Figure 7.

```

|0:Promotion Set:Burger          $14.95
|1:Promotion Set:Chicken Chop    $11.95
|2:Promotion Set:Pork Chop       $12.95
|3:Promotion Set:Lamb Chop       $14.95
|4:Promotion Set:Spaghetti       $10.95

Which item?
10
Item selected out of range

```

Figure 7: Error message: “Item selected out of range”

#### 4.2.4 Creating Reservation Beyond Table Capacity

Case for when a customer creates a reservation for more than 10 people in a single table. Currently, the restaurant tables can only hold up to 10 people. An error message will be shown as seen in Figure 8.

```

===== Please choose one =====
|1. Edit Menu Item
|2. Create Order
|3. Edit Order
|4. View Order
|5. Create Reservation Booking
|6. Check/Remove Reservation Booking
|7. Check Table Availability
|8. Print Order Invoice
|9. Close Sales
|10. Print Sales Revenue Report By Period(by day)
|11. Add/Remove Employee
|12. Exit
=====
5
How many people?
11
No table available for no. of people

```

Figure 8: Error message: “No table available for no. of people”