**SCHOOL OF COMPUTER SCIENCE & ENGINEERING NANYANG TECHNOLOGICAL UNIVERSITY**

**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 honoured 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 | Lab Group | Signature/Date |
| --- | --- | --- | --- |
| Cheng Chang Wei | CZ2002 | SS2 |  |
| Chee Mei Qi | CZ2002 | SS2 |  |
| Goh Zheng Ying | CZ2002 | SS2 |  |
| Kim Minhwan | CZ2002 | SS2 |  |
| Chen Hongpo | CZ2002 | SS2 |  |

**DESIGN CONSIDERATIONS**

**Approach taken:**

Our team uses the features of the Object-Oriented Model to design our program. The 4 features of Object-Oriented Design and Programming (OODP) are namely:

1. Abstraction
2. Encapsulation/ Information-Hiding
3. Inheritance
4. Polymorphism

Combining these features of OODP with the S.O.L.I.D principles, we can achieve **loose coupling and high cohesion** with our code, enabling the code to be easily **reused**, **extended**, and **maintained**.

Our program has utilised the following features of the Object-Oriented Model:

1. Abstraction:

By creating objects of a class with their essential attributes and behaviour(methods), we are utilising abstraction.

1. Encapsulation / Information Hiding:

We utilise access modifiers to protect an object’s *private* data (Encapsulation). For example, object attributes are set to private, which cannot be accessed outside the class. If we want to modify/access such data, we can do so via *public* getter and setter methods. For example, the *loadReservation()* method is a private method, which prevents usage of this method from other classes.

Furthermore, our code uses object composition (delegation), which is a form of information hiding. For example, our *RestaurantApp* class needs to make a new reservation. The *RestaurantApp* just needs to use the *ReservationBoundary* class to call its method and a reservation will be made. *RestaurantApp* does not know how the reservation was made. Such implementation helps to reduce programming complexity.

Why does our code (currently) not utilise Inheritance and Polymorphism?

The purpose of Inheritance and Polymorphism is to reduce the amount of code you have to write by reusing code from an existing class. However, since all our current classes are drastically different and performing very different functions, it is not logical to utilise Inheritance and Polymorphism since we are unable to reuse code. However, our classes remain open to extension if more added functionality needs to be added to the classes.

**Principles used:**

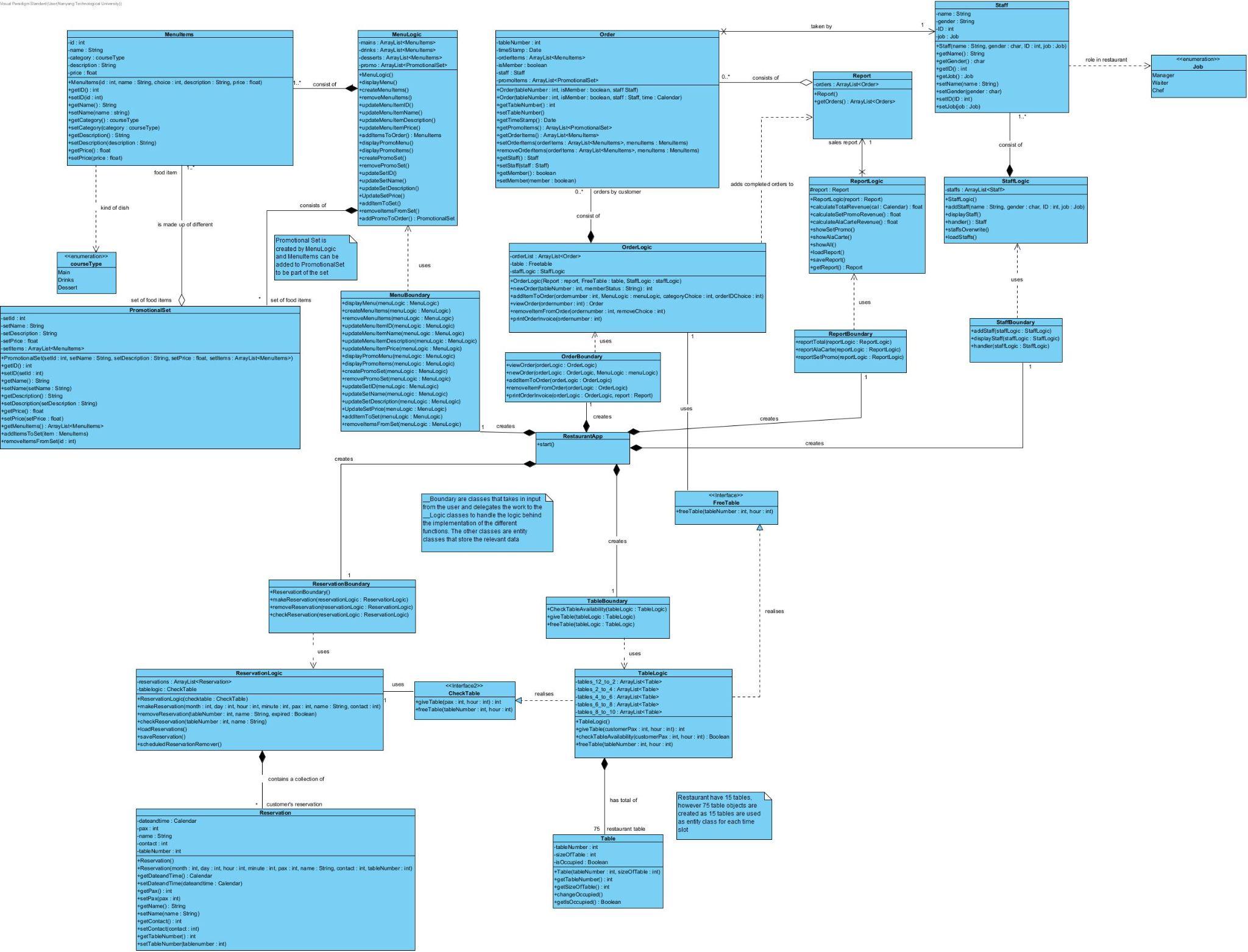
1. Single Responsibility Principle (SRP)
   1. If a class has more than one responsibility, then the responsibilities become coupled.
   2. Showcased by our classes performing a single responsibility, e.g., ReservationBoundary is responsible for just taking in the inputs from the user while ReservationLogic is responsible for managing the reservation objects and lastly Reservation, which is responsible for storing information as an entity class. Each class has their own clearly defined responsibilities and therefore, there is only one reason for the class to change.
2. Integration Segregation Principle
   1. Classes should not depend on interfaces that they do not use. Interfaces with many functions should be avoided.
   2. Demonstrated by our interface CheckTable which has only two functions which are both used in the ReservationLogic class. The interface is kept small and the class does not depend on any interfaces that they do not use.
3. Dependency Injection Principle
   1. High level modules should not depend on low level modules. Both should depend on abstractions. Details should depend on abstraction.
   2. Demonstrated by how ReservationLogic has a dependency on the interface CheckTable (an abstraction). The TableLogic class then implements the CheckTable interface. The benefit is that when the TableLogic class is changed, the ReservationLogic class will not be affected by the change since it is depending on the interface (which is abstract and will not change) instead of the TableLogic class.

Why does our code not use the Open-Closed and Liskov Substitution Principles?

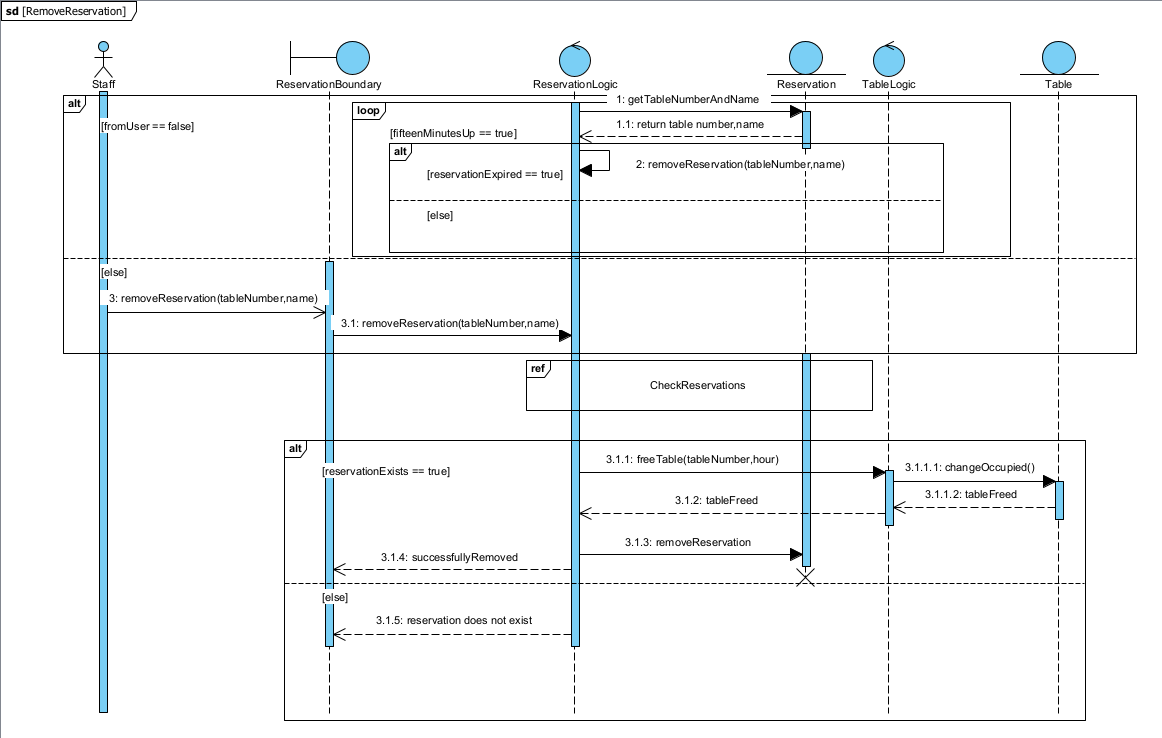
It is due to the absence of Inheritance and Polymorphism in our code due to the reasons mentioned above.

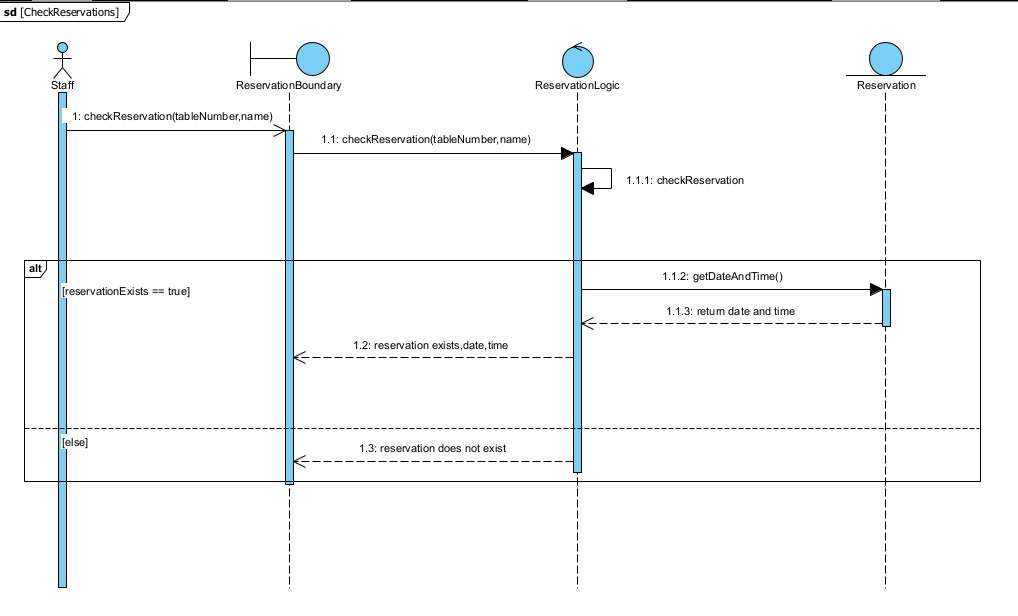
**Assumptions made:**

* No walk-in for restaurant
* Reservations are in blocks of two hours.
* Reservations can only be done one day in advance.
* Total number of tables in the restaurant is fixed.
* Customers have to order before their reservation is removed.
* Customers will not have the same name.

**CLASS DIAGRAMS**

**SEQUENCE DIAGRAMS**

****

****

**TESTING**

| Test Cases | Result (Screen Captures) |
| --- | --- |
| Display Menu read from text file | |
| Category: Menu  Function: Display Menu  Menu to display: Main |  |
| Category: Menu  Function: Display Menu  Menu to display: Dessert |  |
| Category: Menu  Function: Display Menu  Menu to display: Drink |  |
| Update Item in Menu + Display Menu | |
| Category: Menu  Function: Update Menu Items  Variable: Change ID  Old Item ID: 0  Category: Main  New Item ID: 13 |  |
| Category: Menu  Function: Update Menu Items  Variable: Change Description  Item ID: 5  Category: Main  New description: Fries that will never turn soggy |  |
| Display Promotion Menu read from text file | |
| Category: Menu  Function: Display Promo Menu |  |
| Update an item on the Promotion Menu + Display Promo Menu | |
| Category: Menu  Function: Update Promo Menu  Variable: Change ID  Old Promotional Set ID: 0  New Set ID: 2 |  |
| Category: Menu  Function: Update Promo Menu  Variable: Change Name  Promotional Set ID: 1  New Set Name: New Year Promotion |  |
| Category: Menu  Function: Update Promo Menu  Variable: Remove item from set  Promotional Set ID: 1  Item ID to remove: 10 |  |
| Table is not available when checking table availability | |
| Category: Table  Function: Check table availability  Number of customers: 15  Hour of reservation: 22 |  |
| Check Reservation but user does not have a reservation | |
| Category: Reservation  Function: Check reservation  Reservation table number: 1  Customer name: John |  |
| Check Reservation but table number is invalid | |
| Category: Reservation  Function: Check reservation  Reservation table number: 16  Customer name: John |  |
| Remove Reservation but reservation is not yet made | |
| Category: Reservation  Function: Remove reservation  Reservation table number: 1  Customer name: John |  |
| Display staff list read from text file | |
| Category: Staff  Function: Display staff |  |
| Add a new staff + print | |
| Category: Staff  Function: Add staff  Name: Gordon Ramsay  Gender: M  Staff ID: 4  Staff job: Chef |  |