CZ2002 OBJECT-ORIENTED DESIGN AND PROGRAMMING

Restaurant Reservation Point-of-Sale System (RRPSS)
BY: SDDP3 Group 5

<u>Declaration of Original Work for CE/CZ2002 Assignment</u>

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	Lab Group	Signature /Date
Thong Jia Li, Janessa (U1822617J)	CZ2002	SDDP3	[illegible] 14/11/21
Zeng Xunyi (U2022509A)	CZ2002	SDDP3	[illegible] 14/11/21
Sydney Teo Wen Xuen (U2021555B)	CZ2002	SDDP3	Sydney 14/11/21
Sng Yi Xuan (U2021009B)	CZ2002	SDDP3	[illegible] 14/11/21
Timothy Chang Zu'En (U2022114A)	CZ2002	SDDP3	[illegible] 14/11/21

SECTION 1: BACKGROUND

1.1 Background Information

Hard Rock Cafe is a restaurant of its own kind, offering a wide range of cuisines and flexible meal combinations. Customers enjoy dishes like Premium Tomato, Extra Premium Lollipops, and Surprise Sets, and enjoy the ever-changing and customised menu even more. The attractive member perks are another one of Hard Rock Cafe's strengths.

Hard Rock Cafe's entry into the market was well-received and met with record-breaking levels of demand in their first quarter of operations, largely due to their top quality cuisine, great ambience and excellent customer service. This has posed significant challenges to the chefs and service crew who struggle to keep up with the surge in customer traffic. A significant portion of time and manpower was previously allocated to manually recording customer orders and attending to bill requests.

To accommodate this overwhelming demand, the restaurant has employed the help of 5 NTU students to develop a **Restaurant Reservation Point of Sale System**. The application serves to computerise the processes of making reservations, recording of orders and displaying of sale records, and is fully operated by restaurant staff.

1.2 Project Requirements

The restaurant would like a system that streamlines processes related to the menu, reservations and orders, and invoice generation, through a console-based application for simplicity.

IMPORTANT LINK TO YOUTUBE VIDEO:

https://www.youtube.com/watch?v=y_m0I0LgOEM

Hi Prof, if you face any difficulties viewing our video or diagrams (Sequence Diagram may be a bit small and unclear), kindly access the Google Drive Folder here to get a better look:

https://drive.google.com/drive/folders/1XN6diIfHbdwmXKQFV2KUcnB2ca4h99A9?usp=sharing

LINK TO OUR GITHUB AND TO CLONE:

https://github.com/XunyiiZ/cz2002-rrpss.git

SECTION 2: DESIGN CONSIDERATIONS

2.1 Overall Approach

Through our Object-Oriented Approach, we hope to achieve the following goals:

1. Ensure the software is easy to maintain and modify, while being able to accommodate new functionalities and the ability to improve future design
2. Accurately reflect relationships and manage dependencies between entities
3. Minimise ripple effect of change on the ecosystem resulting from changes in dependents

As such, our design considerations will prioritise **practical features and user-friendliness**. For example, our **reservation interface** prompts and guides users (staff) systematically with only the essential details that are required to create a reservation booking, and manage occupancy of tables. The **Menu Management System** also allows restaurant management to keep the menu relevant and updated with Seasonal Menus to attract customers. We take n-tier architectural style where higher layers can make use of lower layers. In our case, the UI classes can make use of Controller classes to interact with the Entity classes.

2.2 OO Concepts

Our application made use of the 4 key OO concepts: **Abstraction, Encapsulation/Information Hiding, Inheritance and Polymorphism**.

Abstraction

Abstraction involves identifying the essential attributes and actions of an object that distinguish it from other objects. This concept can be observed from our application where the classes created have their own state, attributes and behaviour. For example, in our ***Reservation*** entity class we identified that the essential characteristics of a reservation should contain details regarding its id, name and contact of the person making the reservation, the number of people the table is being received for, the table id which is being reserved as well as the appointment date and time. This provides crisply defined boundaries relative to the perspective of the user

For example, the abstractController is an abstract class and the implementation of load() and save() methods are provided by the derived classes InvoiceController, MemberController, MenuController, OrderController, and ReservationController.

Encapsulation/Information Hiding

Encapsulation is observed commonly in our application where the details or implementation of a class are hidden from other classes in order to protect an object's private data. These private attributes can only be accessed by other classes via *get* methods. For example, the ***Member*** class has attributes "***name***", "***isMember***" and "***contact***" which are set to private to prevent other classes from accessing these private data.

Inheritance

Inheritance allows new classes to be derived from existing classes by absorbing their attributes and behaviours, and adding new capabilities in the new classes. This enables **code reuse** and greatly reduces programming effort in implementing new classes. For our application, ***AlaCarte*** and ***Set*** classes inherit from ***MenuItem***. This allows attributes and methods common to both ***AlaCarte*** and ***Set*** to be inherited from parent class ***MenuItem*** without needing to repeat code.

Polymorphism

Polymorphism is the ability for a method to take different forms based on the parameters passed to it. This can be done by **method overloading or overriding**. We demonstrate this through the use of the controller classes which override the ***load*** and ***save*** methods of the ***Abstract Controller*** class that handles the read, write, load and save functionalities of handling text files.

2.3 Design Principles

We identify three types of classes needed: Entity, Boundary and Control. Boundary Classes handle input from the ***Staff*** user and output generated from functions. Control classes are needed to handle function logic and application flow, and are needed as we have many entities interacting with each other within the system. We create their respective control classes according to functional requirements. Overall, this ensures proper delegation of responsibilities among classes: staff users will only interact with boundaries, boundaries interact with controllers and other controllers, and controllers interact with entities.

For our application, we aim to adhere to the SOLID design principles.

Firstly, the **Single Responsibility Principle,** where each class has only one responsibility. This lowers coupling and ensures there will not be more than one reason for changes to the class. This principle is continuously applied in the UI and Controller classes to reduce functional overlaps. Figure 2.1 lists some examples of the single responsibility for some classes in our application:

Class	**Responsibility**
MainUI	Main User Interface which connects to all other Interfaces
ReservationUI	Interface that facilitates interaction between Staff and Reservations
ReservationController	Controls logic for incoming booking requests and responsible for creating, checking and removing reservations
Reservation	Entity containing information and attributes of the reservation
TableUI	Interface that facilitates interaction between Staff and Tables
TableController	Controls logic for table information access and responsible for retrieving relevant table information such as ID and Occupancy
Table	Entity containing information and attributes of the table

Figure 2.1 Single Responsibility of Classes

Secondly, the **Open-Closed Principle** states that modules should be open for extension but closed for modification. We adhere to this through the **use of abstraction in designing the *AbstractController*** class, which deals with the handling of text files, which is needed in almost all of our controller classes. The AbstractController class implements "read" and "write" methods which are open for extension and closed for modification by any controller classes as this method is not to be changed between controllers. However, methods like "load" and "save" are abstract methods which require the respective controllers to implement their respective logic in order for the class to be a concrete class.

Thirdly, the **Liskov Substitution Principle** states that subtypes must be substitutable for their base types, since the pre and post conditions of the subclass method would still be met with no difference to the base type. This can be seen from the inheritance of ***Set*** and ***AlaCarte*** from ***MenuItem***. For example, when we call the getPrice() method of ***MenuItem***, either the ***Set*** or ***AlaCarte*** return their prices respectively. In this case, Set returns the sum of the prices of its AlaCarte items multiplied by the set discount, which is expected of the class and hence it functions properly as per its super class and does not bring any trouble its superclass does not.

Fourth, the **Interface Segregation Principle** advocates the use of various client-specific interfaces over one general purpose interface. This is to prevent classes from depending on interfaces they will not use and implementing certain methods the class does not use. In our case, we did not see the need to use any interfaces due to the lack of abstract methods involved.
Lastly, the **Dependency Injection Principle** enforces that high level modules should not depend on lower-level modules. To achieve this, abstraction facilitates the dependencies between modules. For example, in the higher-level module of the OrderUI class, the staff just needs to know the abstraction: “Create Order”, “View Order Items”, “Add Order Item to Order”, “Remove Order Item from Order” and “Display all orders”. The lower module of Order Controller depends on these abstractions to implement the respective logic to handle its functionalities.
We applied the relevant concepts associated with each principle in our design considerations to construct the **UML Class Diagram** in *Section 3*.

2.4 Data Structure

For file IO, we read and write the objects to the text file (.txt). This is preferred over the .dat file as the information saved in the file is more readable.

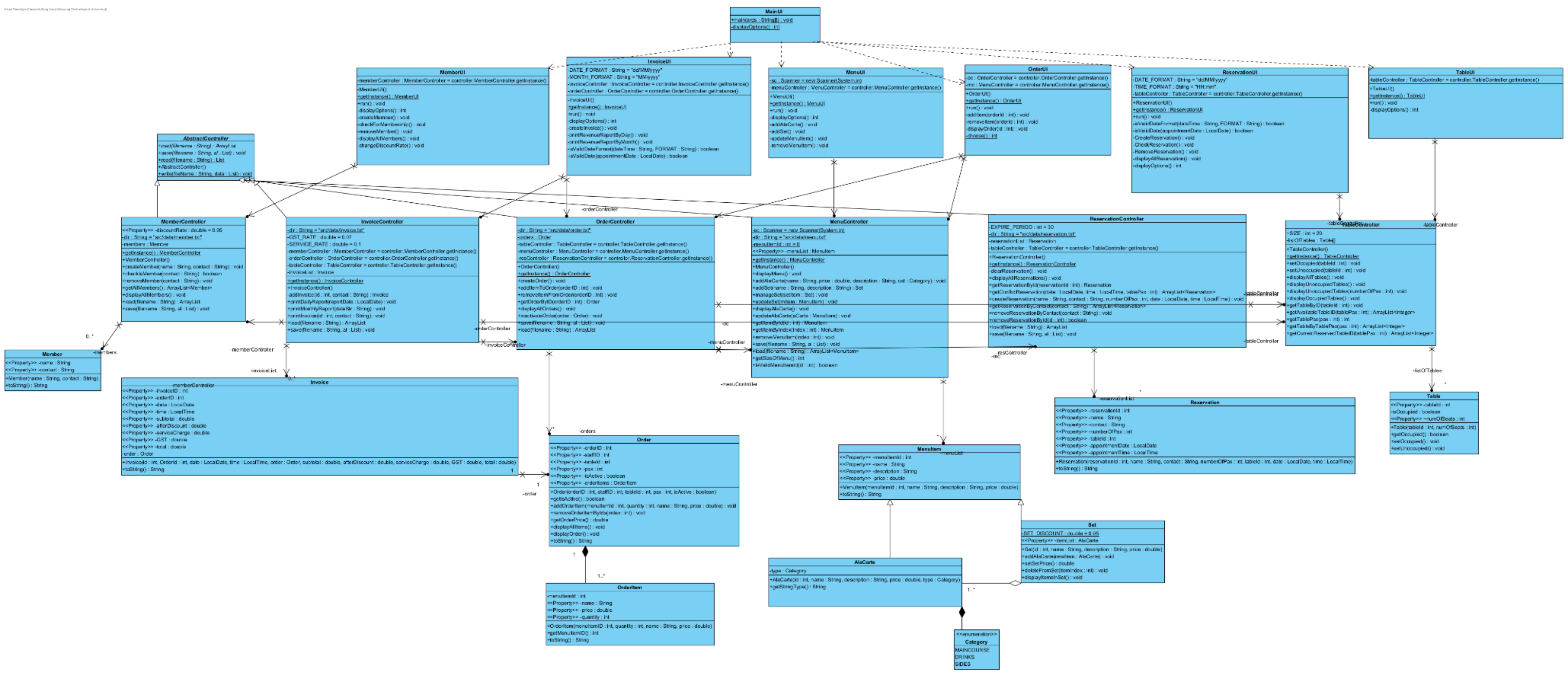
2.5 Assumptions

While designing our application, these are the assumptions we made:

- The system saves only the name and contact number of customers who are members. The details of non-member customers would not be saved in the system.
- Reservations must be made at least 1 day in advance.
- Reservations will be expired after 30 minutes of the appointment time
- After checking in and making an order, the reservation will be removed
- Each reservation and meal lasts 2 hours
- Once an AlaCarte item has been removed from the menu list, any set that contains this item will also be removed as the set is no longer created as it was intended
- Once an order is paid for (invoice printed), the order can no longer be changed or updated
- A Set must contain at least 2 AlaCarte items
- The price of a Set is calculated automatically based on the discounted cost of all the ala carte items in the Set. This price cannot be set manually.

SECTION 3: DETAILED UML CLASS DIAGRAM

3.1 Class Diagram Implementation



3.2 Explanation for Class Diagram

The class diagram consists of the following:

Boundary/UI: InvoiceUI, MainUI, MemberUI, OrderUI, ReservationUI, TableUI, MenuUI

The boundary class is the interface that interacts with the staff of the restaurant.

Control: AbstractController, InvoiceController, MemberController, OrderController, ReservationController, TableController, MenuController

Control classes are objects that mediate between boundaries and entities. They orchestrate the execution of commands coming from the boundary. In the control classes, we implement the logic required to manage the various elements and their interactions.

Entities: AlaCarte, Invoice, Member, MenuItem, Order, OrderItem, Reservation, Set, Table

Entities are objects representing system data.

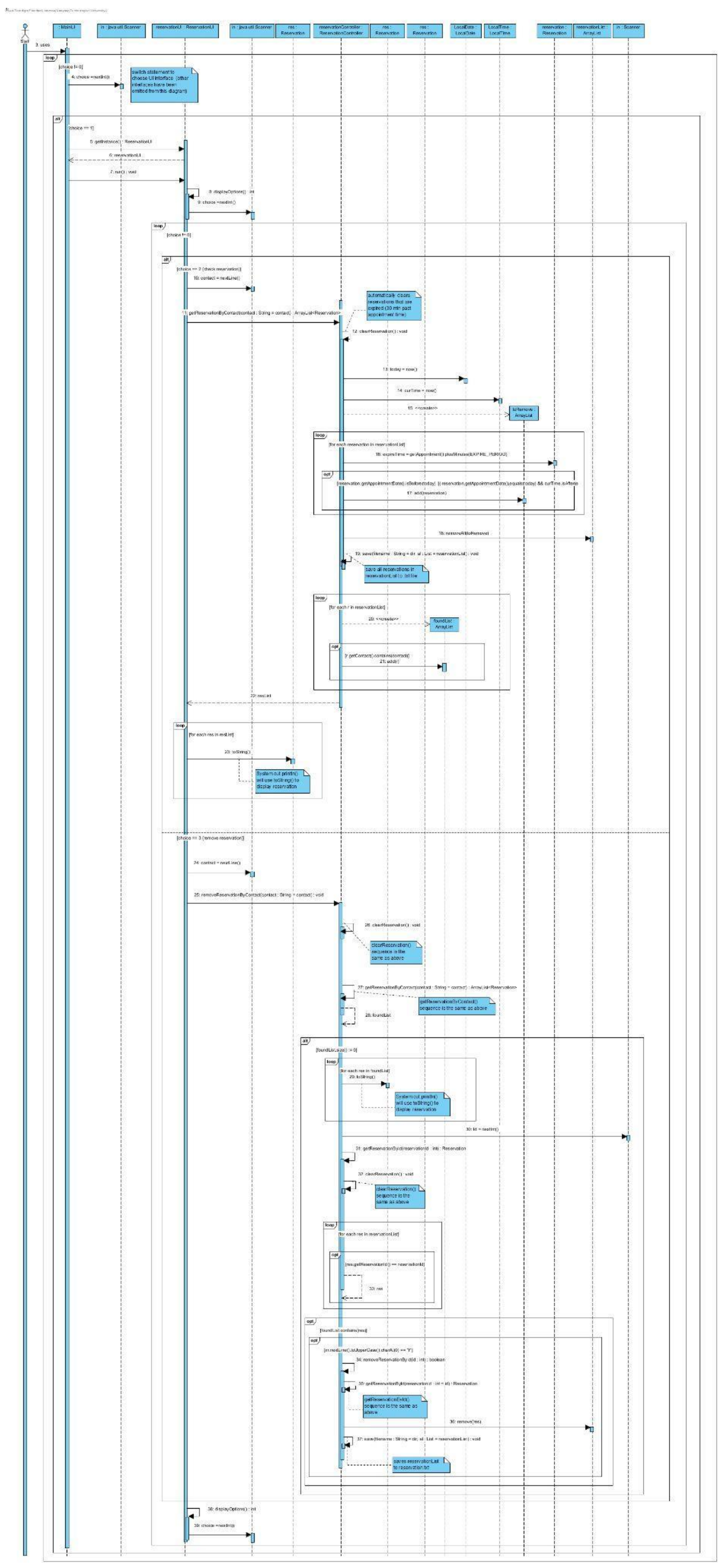
Points to Note:

1. Actors or users (Manager/ Staff etc) can only interact with boundary objects
2. Boundary objects can only interact with controllers and actors
3. Controllers can talk to boundary and entity objects and other controllers, but not to actors
4. Entity objects can only interact with controllers

In general, there will be a MainUI that shares a dependency relationship with the rest of the UI. This is because just as the name suggests, they are dependent on each other. If we make a change to any of the sub-UI, then the MainUI will also get affected by the change. Similarly, each UI also shares a dependency relationship with its respective controllers.

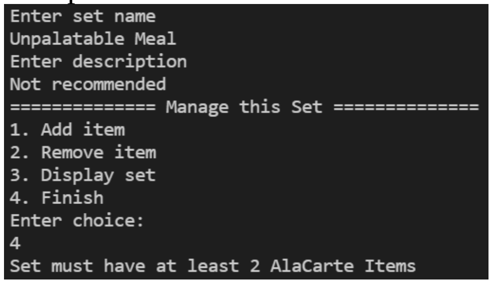
SECTION 4: UML SEQUENCE DIAGRAM

4.1 Sequence Diagram Implementation for Checking/Removing of Reservation Booking



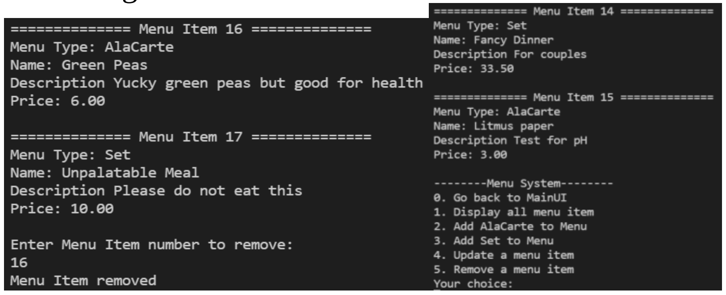
SECTION 5: TESTING

Add promotional item with < 2 ala carte items



```
Enter set name
Unpalatable Meal
Enter description
Not recommended
============== Manage this Set ==============
1. Add item
2. Remove item
3. Display set
4. Finish
Enter choice:
4
Set must have at least 2 AlaCarte Items
```

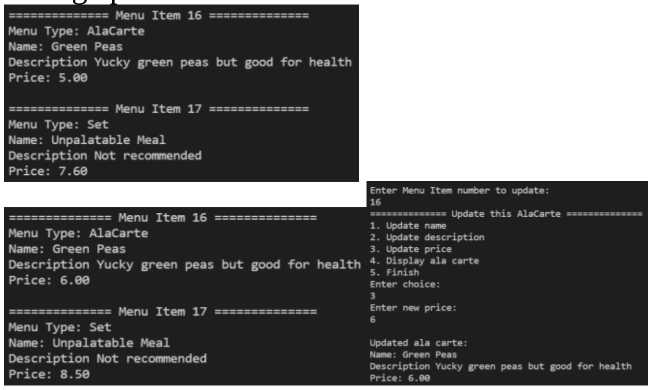
Removing Ala Carte item also removes Set



```
============== Menu Item 16 ==============
Menu Type: AlaCarte
Name: Green Peas
Description Yucky green peas but good for health
Price: 6.00

============== Menu Item 17 ==============
Menu Type: Set
Name: Unpalatable Meal
Description Please do not eat this
Price: 10.00

Enter Menu Item number to remove:
16
Menu Item removed
```

```
============== Menu Item 14 ==============
Menu Type: Set
Name: Fancy Dinner
Description For couples
Price: 33.50

============== Menu Item 15 ==============
Menu Type: AlaCarte
Name: Litmus paper
Description Test for pH
Price: 3.00

--------Menu System--------
0. Go back to MainUI
1. Display all menu item
2. Add AlaCarte to Menu
3. Add Set to Menu
4. Update a menu item
5. Remove a menu item
Your choice:
```

Change price of ala carte

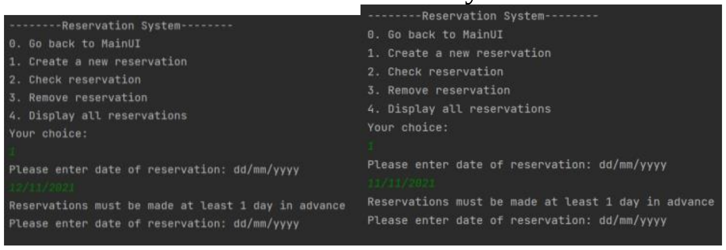


```
============== Menu Item 16 ==============
Menu Type: AlaCarte
Name: Green Peas
Description Yucky green peas but good for health
Price: 5.00

============== Menu Item 17 ==============
Menu Type: Set
Name: Unpalatable Meal
Description Not recommended
Price: 7.60
```

```
Enter Menu Item number to update:
16
============== Update this AlaCarte ==============
1. Update name
2. Update description
3. Update price
4. Display ala carte
5. Finish
Enter choice:
3
Enter new price:
6

Updated ala carte:
Name: Green Peas
Description Yucky green peas but good for health
Price: 6.00
```

```
============== Menu Item 16 ==============
Menu Type: AlaCarte
Name: Green Peas
Description Yucky green peas but good for health
Price: 6.00

============== Menu Item 17 ==============
Menu Type: Set
Name: Unpalatable Meal
Description Not recommended
Price: 8.50
```

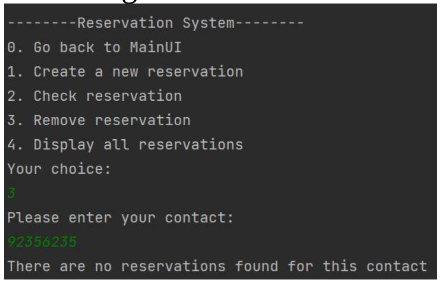
Create Reservation for the Same Day



```
--------Reservation System--------
0. Go back to MainUI
1. Create a new reservation
2. Check reservation
3. Remove reservation
4. Display all reservations
Your choice:
1
Please enter date of reservation: dd/mm/yyyy
12/11/2021
Reservations must be made at least 1 day in advance
Please enter date of reservation: dd/mm/yyyy
```

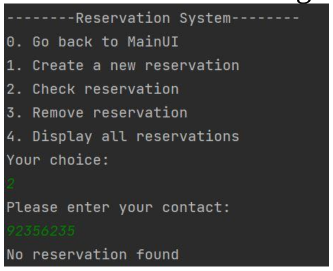
```
--------Reservation System--------
0. Go back to MainUI
1. Create a new reservation
2. Check reservation
3. Remove reservation
4. Display all reservations
Your choice:
1
Please enter date of reservation: dd/mm/yyyy
11/11/2021
Reservations must be made at least 1 day in advance
Please enter date of reservation: dd/mm/yyyy
```

Removing a Reservation that does not exist



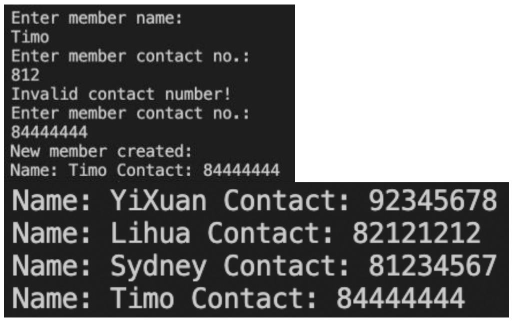
```
--------Reservation System--------
0. Go back to MainUI
1. Create a new reservation
2. Check reservation
3. Remove reservation
4. Display all reservations
Your choice:
3
Please enter your contact:
92356235
There are no reservations found for this contact
```

Check for Non-Existing Reservation



```
--------Reservation System--------
0. Go back to MainUI
1. Create a new reservation
2. Check reservation
3. Remove reservation
4. Display all reservations
Your choice:
2
Please enter your contact:
92356235
No reservation found
```

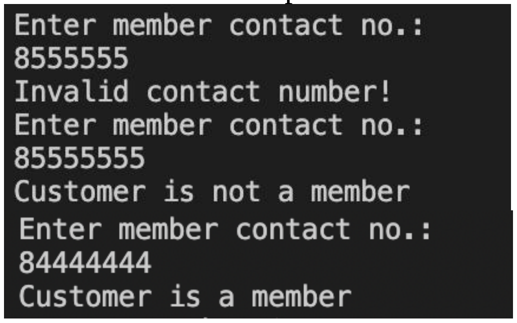
Create New Member / Check Membership



```
Enter member name:
Timo
Enter member contact no.:
812
Invalid contact number!
Enter member contact no.:
84444444
New member created:
Name: Timo Contact: 84444444
```

```
Name: YiXuan Contact: 92345678
Name: Lihua Contact: 82121212
Name: Sydney Contact: 81234567
Name: Timo Contact: 84444444
```

Check for Membership

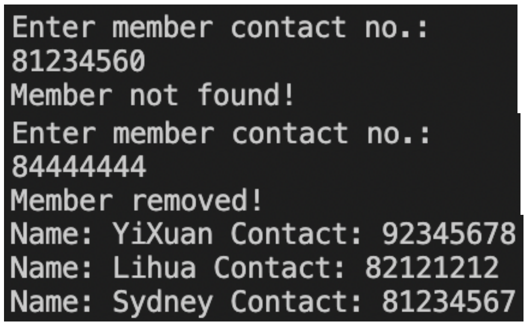


```
Enter member contact no.:
8555555
Invalid contact number!
Enter member contact no.:
85555555
Customer is not a member
```

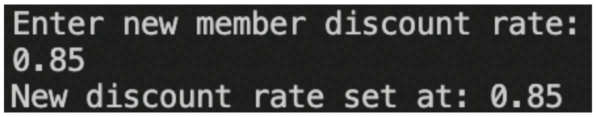
```
Enter member contact no.:
84444444
Customer is a member
```

Remove Member / Check Membership

Change discount rate



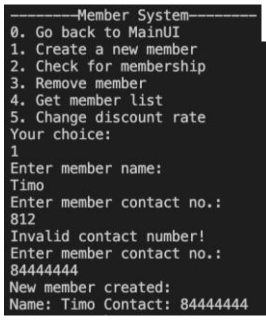
```
Enter member contact no.:
81234560
Member not found!
Enter member contact no.:
84444444
Member removed!
Name: YiXuan Contact: 92345678
Name: Lihua Contact: 82121212
Name: Sydney Contact: 81234567
```



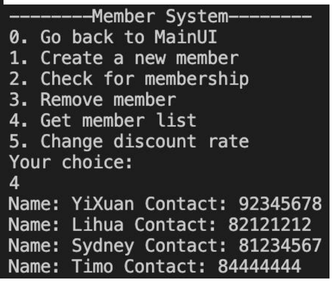
```
Enter new member discount rate:
0.85
New discount rate set at: 0.85
```

APPENDIX

1. Create new member / Get member list

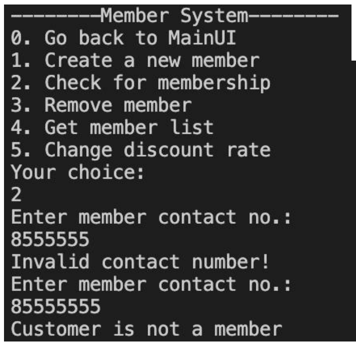


```
--------Member System--------
0. Go back to MainUI
1. Create a new member
2. Check for membership
3. Remove member
4. Get member list
5. Change discount rate
Your choice:
1
Enter member name:
Timo
Enter member contact no.:
812
Invalid contact number!
Enter member contact no.:
84444444
New member created:
Name: Timo Contact: 84444444
```

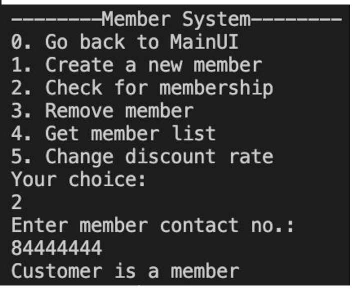


```
--------Member System--------
0. Go back to MainUI
1. Create a new member
2. Check for membership
3. Remove member
4. Get member list
5. Change discount rate
Your choice:
4
Name: YiXuan Contact: 92345678
Name: Lihua Contact: 82121212
Name: Sydney Contact: 81234567
Name: Timo Contact: 84444444
```

2. Check for membership

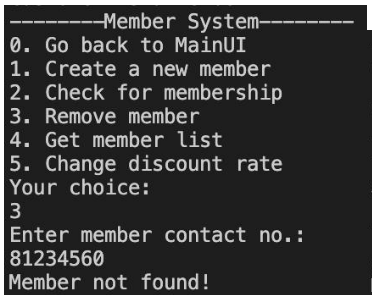


```
--------Member System--------
0. Go back to MainUI
1. Create a new member
2. Check for membership
3. Remove member
4. Get member list
5. Change discount rate
Your choice:
2
Enter member contact no.:
8555555
Invalid contact number!
Enter member contact no.:
85555555
Customer is not a member
```

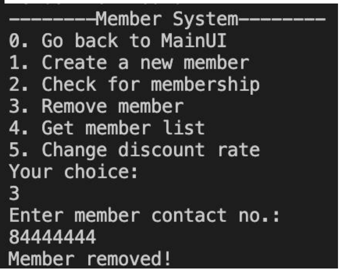


```
--------Member System--------
0. Go back to MainUI
1. Create a new member
2. Check for membership
3. Remove member
4. Get member list
5. Change discount rate
Your choice:
2
Enter member contact no.:
84444444
Customer is a member
```

3. Remove member

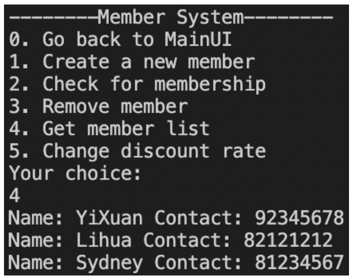


```
--------Member System--------
0. Go back to MainUI
1. Create a new member
2. Check for membership
3. Remove member
4. Get member list
5. Change discount rate
Your choice:
3
Enter member contact no.:
81234560
Member not found!
```



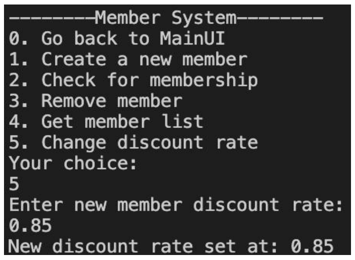
```
--------Member System--------
0. Go back to MainUI
1. Create a new member
2. Check for membership
3. Remove member
4. Get member list
5. Change discount rate
Your choice:
3
Enter member contact no.:
84444444
Member removed!
```

4. Get member list



```
--------Member System--------
0. Go back to MainUI
1. Create a new member
2. Check for membership
3. Remove member
4. Get member list
5. Change discount rate
Your choice:
4
Name: YiXuan Contact: 92345678
Name: Lihua Contact: 82121212
Name: Sydney Contact: 81234567
```

5. Change discount rate



```
--------Member System--------
0. Go back to MainUI
1. Create a new member
2. Check for membership
3. Remove member
4. Get member list
5. Change discount rate
Your choice:
5
Enter new member discount rate:
0.85
New discount rate set at: 0.85
```