


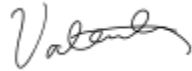


Declaration of Original Work for CE/CZ2002 Assignment

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
Chen Xingyu	CZ2002	DSAI1	 14/11/2021
Jacintha Wee Yun Yi	CZ2002	DSAI1	 14/11/2021
Li Siyi	CZ2002	DSAI1	 14/11/2021
Valencia Lie	CZ2002	DSAI1	 14/11/2021

Youtube link: <https://youtu.be/gFmUuCfeAFI>

1. **Design considerations:**

S- Single Responsibility Principle (SRP)

Single Responsibility Principle (SRP) is clearly demonstrated in our work as none of our classes have multiple responsibilities. For example, our OrderManager class assumes the responsibility of controlling the Order class only (which is our Entity class) while our ReservationManager is responsible for controlling the Reservation class only.

Moreover, we also have boundary classes that assume the sole responsibility of taking in the user's input related to its Entity class, which can then be called by our controller classes.

For example, in our ItemUI class, we take user's input on the details of the item they want to add inside the menu and we pass those attributes to the ItemController class to actually create the item itself.

This means that none of the classes have multiple responsibilities and instead assumes only one important role.

O- Open-Closed Principle (OCP)

Open-Closed Principle (OCP) is clearly demonstrated in our work through the use of abstract and concrete classes. For example, our Food class is an abstract class and our Item and Package classes inherit from it. If we want to make changes to certain methods inherited from the Food superclass within the Item and Package subclasses, such as getName and getPrice methods, we can do so without affecting any changes to the getName and getPrice methods within Food. Hence, we are able to make changes to what the subclasses Item and Package do, without changing the source code of the superclass

Food, which is in agreement with the OCP. This also means that if we ever want to create a new class that has similar attributes and methods to the Item and Package classes, we can basically just let it inherit the Food class, without ever changing our Food class.

L- Liskov Substitution Principle (LSP)

One of the conditions of LSP states that a user of a base class should continue to function properly if a derivative of that base class is passed to it. This principle is demonstrated in our OrderManager class. For example, in the addOrder method, one of the parameters is defined as an object food from the Food superclass. When we call this method in Restaurant class, we can pass in either the Item subclass' object or the Package subclass' object into the Food superclass argument and our program will add the Item or Package order accordingly.

D- Dependency Injection Principle (DIP)

The Dependency Injection Problem states that abstraction should not depend upon details but details should depend upon abstraction. This is demonstrated with how our abstract class Food has abstract methods that are explicitly defined by our subclasses Item and Package. Hence the bulk of implementation is done by lower modules i.e. subclasses instead of the superclass.

Others - Encapsulation

We also demonstrated the OO principle encapsulation through sectioning our classes into 3 main class stereotypes: Boundary, Entity and Controller. By doing so, the main functionalities of our working program (what goes “under the hood”) that are coded under the Controller classes will not be visible to the user, who only interacts with the Boundary class. In turn, the Boundary and Controller classes will interact with each other; the user will never directly interact with the Controller classes. Moreover, objects from the Boundary and Controller classes are initialised in the Restaurant class, hence

these objects are not accessible to the user, who will then be automatically barred editing rights to our program. This further enhances the robustness of our program and ensures that any changes made are done by the creators only.

Others - Polymorphism

Polymorphism is demonstrated in the OrderManager class, whereby the getName and getPrice methods taken from the Food superclass are overridden by the subclasses Item and Package. Hence, the superclass methods from Food will be overridden by methods defined in any new subclasses derived from it. So whenever we add new subclasses derived from Food, we do not need to change the source code in the OrderManager class.

Possible Extension of new features - 1) Members' Birthday

We can offer members more discounts during their birthday / birthday month and send a message to them on the discount to be offered. We can just add the attribute 'birthday' to the Member class and request the member to enter his/her birthday when the addMember method is called in the MemberManager class. Every first day of the month, our program will send a notification to all members whose birthdays fall within that month. The printInvoice method within OrderManager will also be changed such that under the if statement for membership, if the member is dining in within his/her birth month, the extra discount will be applied. We can implement the changes quickly and efficiently without having to change a lot of our source codes because of our implemented design.

Possible Extension of new features - 2) Customisation of Order

Some customers might have food allergies, for example an allergy to peanuts, and would wish to specify in their order for the cook not to include peanuts in their food. Another variable 'specialNote' can be added to the addOrder method in the OrderManager class for staff to input specific instructions from customers in their order. This will allow for small customisation of food according to each customer's needs and preferences before the order is placed, without us having to change a lot of our source codes.

For higher resolution, please go to this link: shorturl.at/afuG1

3. UML Sequence Diagram

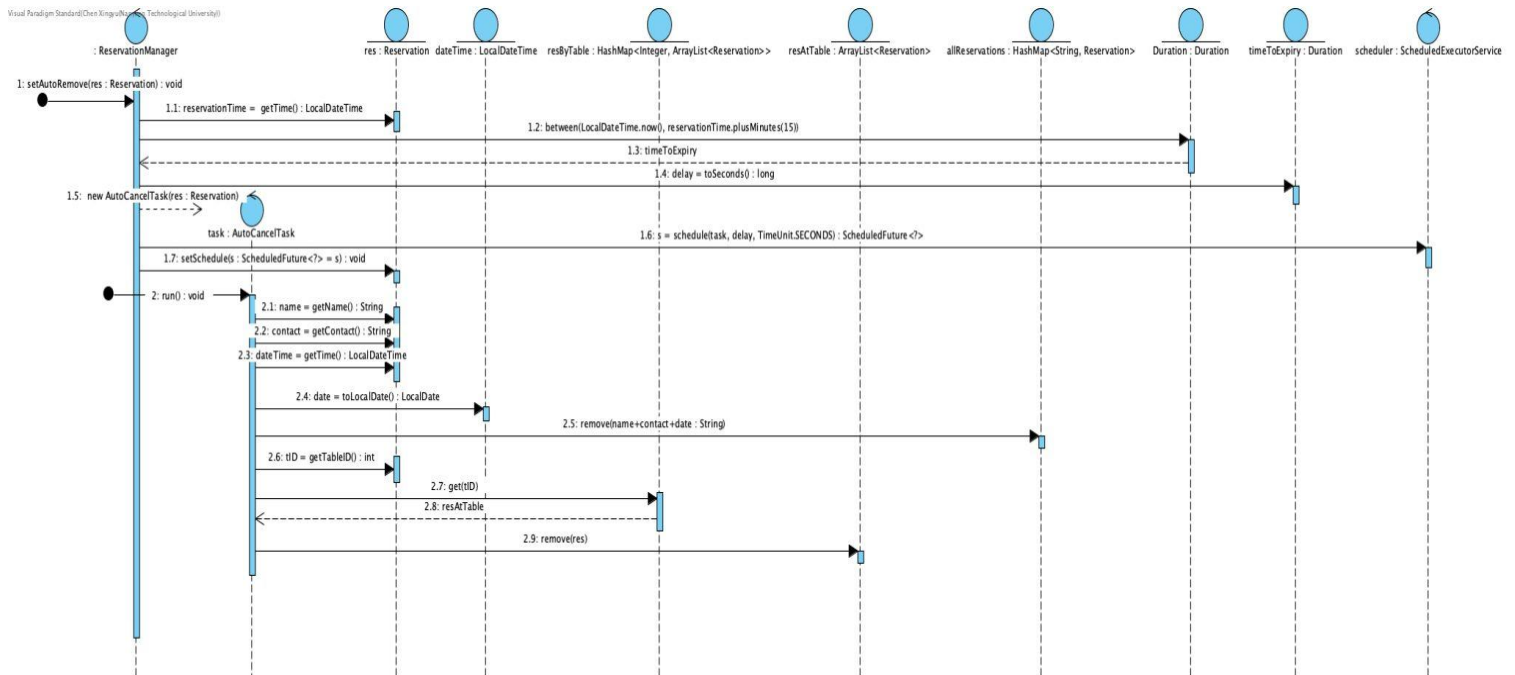
For higher resolution, please go to this link: shorturl.at/afuG1

a) Auto remove

Below is the sequence diagram to remove a reservation upon expiry (15 minutes after reservation time). The steps are:

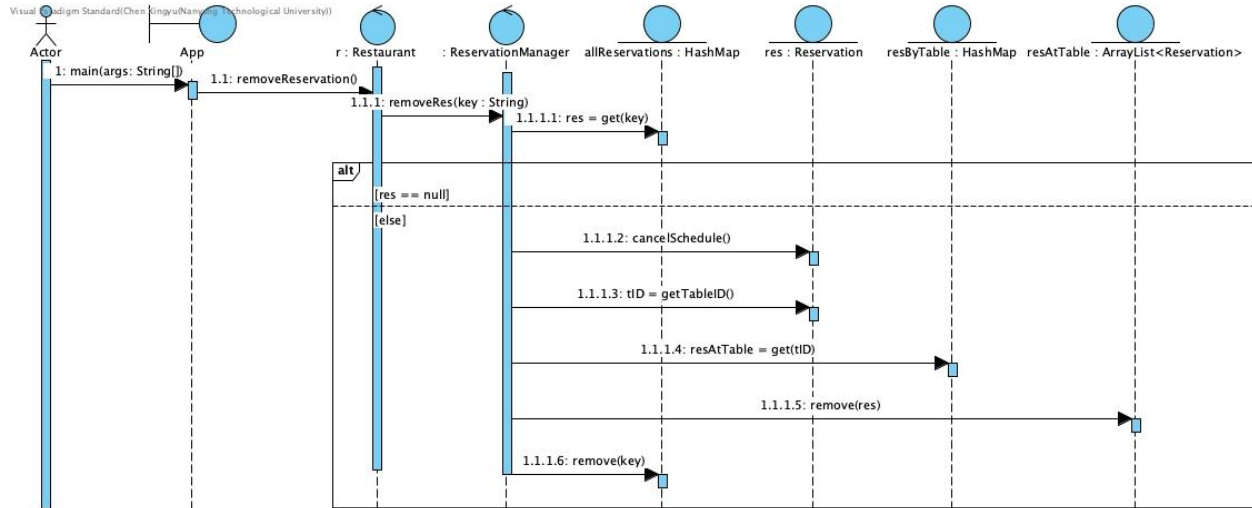
When the user makes a new reservation, we will call `setAutoRemove(Reservation res)` in `makeReservation()` method to set up a scheduled task

“`AutoCancelTask(res)`” by passing the task to `scheduler.schedule(task, delay, TimeUnit.seconds)`. The task is responsible for removing the reservation upon expiry. This `setAutoRemove(Reservation res)` method is shown as 1.x in sequence diagram a. Then when it comes to the expiry time, the system will execute the task by calling the `AutoCancelTask.run()` and inside `run()` method, it will remove the reservation. The `run()` method is shown as 2.x in sequence diagram a.



b) Manually remove

This method is used when the reservation is not expired yet and the user wants to remove the reservation manually.



4. Assumptions of program

We assume the reservation duration is 2 hours and the walk-in customers also take at most 2 hours to finish when we check table availability. We also don't allow customers with the same name and contact number to make another reservation on the same day. We can only make packages with items that already exist in the menu. Prices for packages can only be lower than the total prices of all items inside. We assume the tax rate is in accordance with Singapore's tax GST and service charge is 17%.

Note: To use the Notification class inside, please import the jar files required and uncomment Notification.send in Restaurant Class (line 206)

5. Screenshots of program

Other edge cases not shown in video:

1. When the price of the package entered is higher than the total price of all the items inside the package, there will be an error encountered because it does not make sense for the price of the promotional package to be higher.

```
Enter the number of your choice: 11
What is the index of the package?
0
What is the name of the package?
pasta and fries
Add items to package! Press c to continue, press q to stop adding items.
c
What is the index of the item?
1
How much quantity of this item?
2
Add items to package! Press c to continue, press q to stop adding items.
q
Invalid input. Please enter either c or q.
c
What is the index of the item?
0
How much quantity of this item?
0
Add items to package! Press c to continue, press q to stop adding items.
q
What is the price of the package?
5
Invalid input. Price cannot be smaller than 0 or higher than the total price of individual items inside the package. Please enter again.
6
Added package successfully.
```

```
*****
Package Index = 1
Package Name = 2 pastas
Original Package Price = 6.0
Discounted Package Price = 5.0

quantity: 2
Item Index = 0
Name = pasta
Description = -
Price = 3.0
Type = MAIN
```

As we can see from above, the package price is \$5 but the original package price (which is the total price of all the items inside the package is \$6)

2. We also handle exceptions when the user entered an available table when creating order because an order can only be made when a table is occupied

```
Enter the number of your choice: 15
Enter table number:
3
This table has not been assigned yet. Please dine in first before creating order.
```


3. We also handle cases when the user enters the wrong StaffID when creating an order because only staffs are allowed to create orders.

```
Enter the number of your choice: 15
Enter table number:
1
Enter staff's name:
Sourav
Enter staff's ID:
1016
Invalid staffID entered
Unable to create order as staff server particulars are wrong or does not exist.
```

4. We also don't allow users to put in an invalid price/quantity for an item/package, for example if the price is <0 or if the price is not of the correct double format.

```
How much quantity of this item?
-2
Invalid input, quantity cannot be smaller than or equals to 0. Please enter again.
How much quantity of this item?
2
```

```
Enter the number of your choice: 8
What is the index of the item?
3
What is the name of the item?
noodles
What is the description of the item?
-
What is the price of the item?
-8
Invalid input. Price cannot be smaller than 0. Please enter again.
What is the price of the item?
10
What is the type of the item? 1: Main, 2: Sides, 3: Beverage, 4: Dessert. Enter either 1,2,3 or 4.
1
Item is added.
```

5. We don't allow for duplicate indexes for item and package.

```
Enter the number of your choice: 8
What is the index of the item?
1
Duplicate index in this menu.
```

6. We don't allow for a package to be empty before creation so once we create a package, it will automatically have at least 1 item.

```
Add items to package! Press c to continue, press q to stop adding items.
q
Package is empty, please add some items before quitting.
```

7. We also handle exceptions when we add items with non-existent indexes into our package or order.

```
Enter the number of your choice: 11
What is the index of the package?
2
What is the name of the package?
2 pastas
Add items to package! Press c to continue, press q to stop adding items.
c
What is the index of the item?
9
Nothing is added into this package because there is no item with this index.
```

8. When we update the price of an item, we also automatically ask the user if they want to change the price of packages that contain that item.

```
Enter the number of your choice: 9
What is the index of the item?
1
What do you want to update? Enter 1 for Index, 2 for Name, 3 for Description, 4 for Price and 5 for Type of food.
4
What is the price of the item?
0.5
Updated successfully
This item is inside package1 pasta and fries please update the price of the package since the total price of each item in the package is higher.
The initial discounted price of this package is 3.0
What is the price of the package?
1
This item is inside package2 pasta please update the price of the package since the total price of each item in the package is higher.
The initial discounted price of this package is 3.0
What is the price of the package?
1
Invalid input. Price cannot be smaller than 0 or higher than the total price of individual items inside the package. Please enter again.
What is the price of the package?
0.5
```