

## **Introduction**

We are tasked to design and develop a Restaurant Reservation and Point of Sale System (RRPSS), an application that will be used by restaurant staff. The RRPSS aims to be an application that improves the workflow and time efficiency of the restaurants' day-to-day operations. It does so by computerizing the processes of managing reservations, orders and the generation of sales reports. It will also have the capability of editing the items on the menu, creating and editing promotional packages, all with ease. With this implementation, the restaurant staff will be able to produce their best performance in providing quality table service without the stress of performing laborious tasks.

## **Design Considerations and OO Concept**

### **Apache Maven Java Project**

We have opted to use an Apache Maven project as Maven allows for external dependencies or external packages to be added or removed in a way that is least intrusive for all project members.

Some external dependencies we are utilizing are Simple Logging Facade for Java (SLF4J) and Google GSON. These external dependencies reduces redundancy by ensuring that our development process can be focused on implementing our business logic and progressing quickly in the development lifecycle instead of constantly "reinventing the wheel"

### **Minimal Configuration Database / File Storage System**

As we are restricted from using SQL based database solutions, we are assuming NoSQL based database solutions such as MongoDB are not allowed too and we are required to come up with our own solution for data storage.

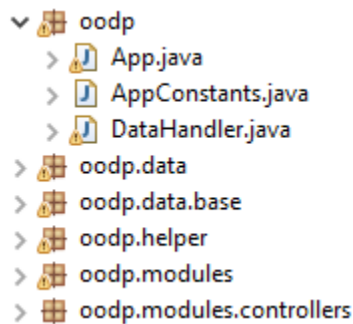
It is imperative to have a robust data access and data storage system that requires minimal interaction while providing maximum control, hence we have developed a data storage solution that requires zero effort from the developers other than inheriting from a 'BaseData' class

This solution relies on a DataHandler class which invokes a series of methods that acts on the classes that are inheriting 'BaseData', with the intention of serializing the superclasses into flat JSON strings through inheritance.

Having a simple and robust system would mean that there is little to no learning curve for developers to utilize the system.

Data access and storage is as simple as writing 1 line to create/retrieve/update/delete data.

### **Clearly Defined Codebase Structure**



```

v oodp
  > App.java
  > AppConstants.java
  > DataHandler.java
  > oodp.data
  > oodp.data.base
  > oodp.helper
  > oodp.modules
  > oodp.modules.controllers

```

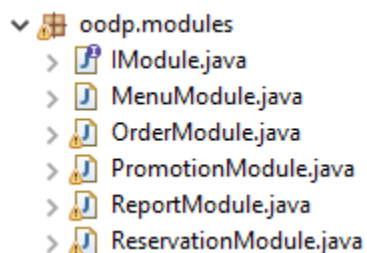
The code structure is clearly defined with boundaries, controllers and entities clearly separated.

This allows for an “easy to learn” codebase implementation as the functionality of each individual class can be inferred from its package path and class name.

Helper classes are introduced to reduce code redundancy and promote clean code alongside error-free code as exceptions are handled by the same method calls.

The helper classes (IOHelper, DataHandler, SeedHelper) are accessible by all modules and unique to none, this allows for maximum code compatibility and reusability. These helper classes contain many instances of overloading to also help maximize their flexibility and readability leading to seamless integration with the rest of the programme.

### **Modular Code Design (Separation of concern)**



```

v oodp.modules
  > IModule.java
  > MenuModule.java
  > OrderModule.java
  > PromotionModule.java
  > ReportModule.java
  > ReservationModule.java

```

The codebase is developed with a structure that minimizes overlaps of code intention. A modular system is used; where each individual work area is separated into their respective module classes.

Each module is clearly defined to have implementations for a specific work area that can be inferred by their class name in a sensible manner.

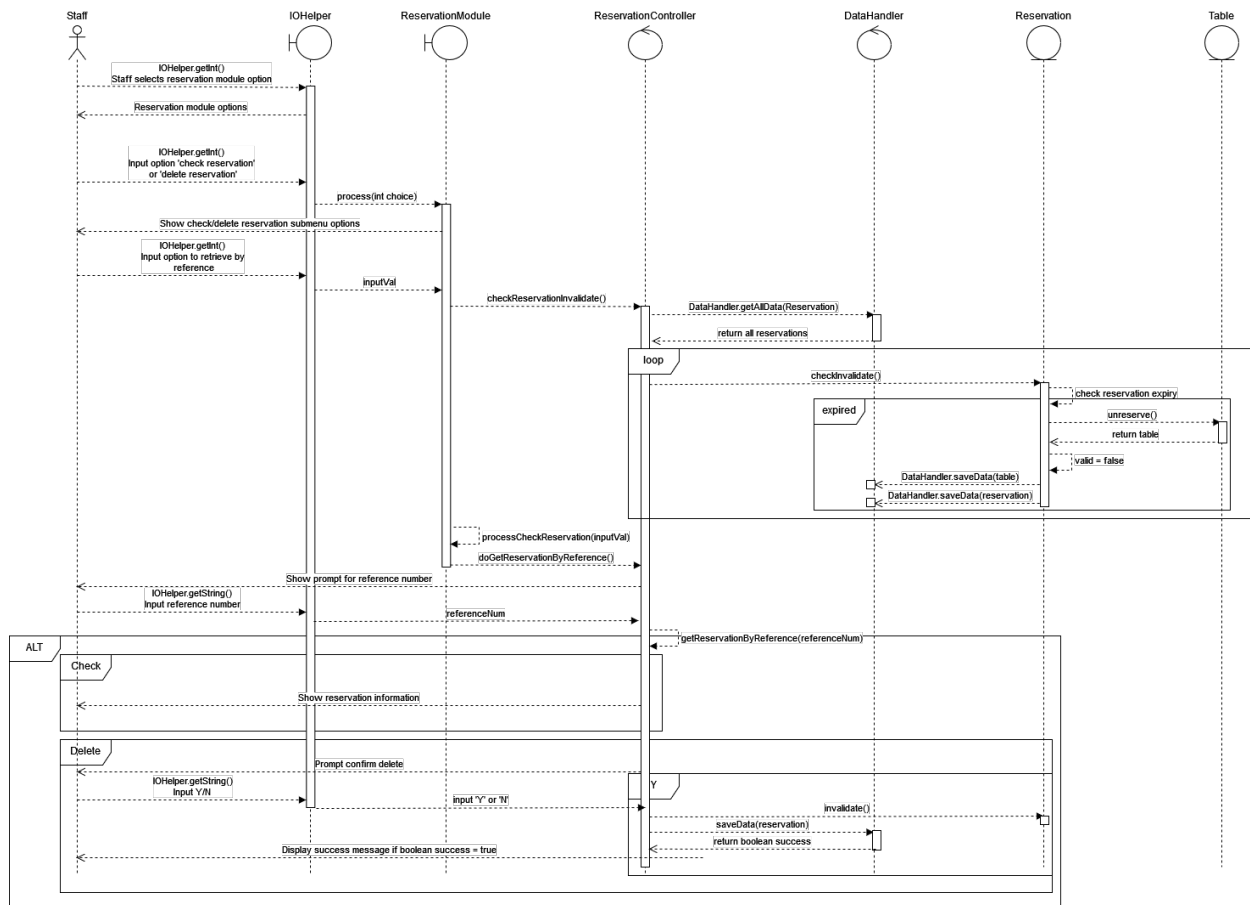
Having a modular code design ensures a clearly defined separation of concern pattern and minimizes code conflicts to improve work efficiency, code cleanliness and to reduce development frustration.

### **Single Responsibility Principle (SRP)**

Each class was created with specific responsibilities in mind so that there is never more than one reason for it to change. By following this principle, our classes will adhere to one functionality, having their data and methods to serve their sole purpose. This results in high cohesion and reliability, which ultimately reduces error.

[illegible]

# Sequence Diagram for Checking/Removing Reservation



## Important/notable parts

### Removal of reservation that are past their expiry time

Before any result is created, retrieved, updated or deleted from the data storage system for processing, ALL reservations in the system are retrieved and iterated. The sole purpose of this iteration is to invalidate all reservations that are past their expiry/decay time (fixed at 5 minutes after Reservation time in AppConstants.java). Hence every statement thereafter are free of reservations that are past their expiry date.

Invalidation of a reservation entry means the reservation's 'valid' flag is set to false, and the update is persisted into the file system.

The general flow of invalidation is

- 1) Reservation flow triggered (check/create/delete reservation) from submenu

- 2) ReservationController.checkReservationInvalidate() is called
  - a) All reservation is retrieved from data store
  - b) Each reservation's checkInvalidate() method is invoked
    - i) If the reservation's 'reservationDecayDateTime' is before the current date time, this means that it has expired and should be invalidated.
    - ii) From (i) if expired, valid is set to false. The related reserved table's unreserve() method is invoked
    - iii) The changes to Table and Reservation are persisted to storage
- 3) The process only continues after the invalidation check has taken place to ensure expired data are not considered valid.

# Screen Captures

## Test Case 1

Making a reservation prior to current date

Please enter customer name: Tom

Please enter customer contact: 91293875

Please enter membership ID (leave blank if not applicable):

Please insert pax count: 1

Please enter date and time (dd/MM/yyyy HHmm)[24 hour clock] : 01/01/1996 1600

The date and time you've entered must be AFTER 14/11/2021 2349

Please enter date and time (dd/MM/yyyy HHmm)[24 hour clock] :

## Result

User is prompted over and over again until a valid input (advanced date and time) is provided.

## Test Case 2

Inserting unexpected value ranges as menu navigation options

Welcome to RRPSS V1.0!

1) - Menu

2) - Promotions

3) - Orders

4) - Reservations

5) - Other Management

Please enter a number from 1 to 5: -8490684096406460454065

[INVALID INPUT] Please enter a number from 1 to 5

1) - Menu

2) - Promotions

3) - Orders

4) - Reservations

5) - Other Management

Please enter a number from 1 to 5: -5.1560

[INVALID INPUT] Please enter a number from 1 to 5

1) - Menu

2) - Promotions

3) - Orders

4) - Reservations

5) - Other Management

Please enter a number from 1 to 5:

## Result

User is informed of the invalid input and program loops without problems

### **Test Case 3**

Inserting blanks, and long string inputs

```
1) - Check Reservation
2) - Check Table Availability
3) - Create Reservation
4) - Delete Reservation
5) - Go Back
Please enter a number from 1 to 5: 3
```

```
Please enter customer name:
[INVALID INPUT] Please enter a string with a minimum length of 1
Please enter customer name: FAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[INVALID INPUT] Please enter a string with a maximum length of 50
Please enter customer name:
```

### **Result**

Prompt is repeated until a valid input is provided

### **Test Case 4**

Deleting the data folder and it's contents

### **Result**

Table data is seeded upon application restart, all other data produced at runtime does not encounter any error.

However, if the program is running and data is deleted, the program might encounter errors from data that is already loaded into memory.

## **Video Demonstration Link**

[https://youtu.be/agjiU6\\_uLBM](https://youtu.be/agjiU6_uLBM)



# Javadoc

All Javadoc files can be found in the {PROJECT}/javadoc folder

OVERVIEW	PACKAGE	CLASS	USE	TREE	DEPRECATED	INDEX	HELP

## Packages

Package
oodp
oodp.data
oodp.data.base
oodp.helper
oodp.modules
oodp.modules.controllers

OVERVIEW	PACKAGE	CLASS	USE	TREE	DEPRECATED	INDEX	HELP
----------	---------	-------	-----	------	------------	-------	------

OVERVIEW	PACKAGE	CLASS	USE	TREE	DEPRECATED	INDEX	HELP
SUMMARY: NESTED   FIELD   CONSTR   METHOD			DETAIL: FIELD   CONSTR   METHOD				

Package oodp.helper

### Class IOHelper

java.lang.Object  
oodp.helper.IOHelper

public class IOHelper  
extends java.lang.Object

IOHelper class that manages most input/output of the program from menu navigation to primitive user input prompting

#### Field Summary

Fields		
Modifier and Type	Field	Description
<code>static int</code>	<code>menu_state</code>	The current menu page to be displayed
<code>static boolean</code>	<code>poll</code>	The program exits if poll is ever set to false
<code>static java.util.Scanner</code>	<code>sc</code>	Scanner singleton

#### Constructor Summary

Constructors	
Constructor	Description
<code>IOHelper()</code>	

#### Method Summary

All Methods	Static Methods	Concrete Methods
Modifier and Type	Method	Description
static java.util.Date	getDateTime(java.lang.String format, boolean repeatUntilSuccess, java.lang.String promptMessage, java.lang.String errorMessage)	Managed method for getting date and time from IO
static	getDateTimeAfter(java.util.Date dtThreshold, java.lang.String format, boolean repeatUntilSuccess,	Managed method for getting date and time AFTER a th

Package oodp.modules.controllers

**Class ReservationController**

java.lang.Object  
oodp.modules.controllers.ReservationController

public class ReservationController  
extends java.lang.Object

Control class between ReservationModule and Reservation Entity

Constructor Summary

Constructors	
Constructor	Description
ReservationController ()	

Method Summary

All Methods	Instance Methods	Concrete Methods	
Modifier and Type	Method	Description	
boolean	assignReservationTable(Reservation reservation)	Assign a table to a reservation based on capacity	
void	checkReservationInvalidate()	Method to loop through all reservations to check and invalidate them if necessary	
Reservation	doGetReservationByReference ()	IO method for getReservationByReference	
void	doListAllReservations ()	IO print to list all valid and active reservations	
Reservation	doReservationByNameAndContact ()	IO method for getReservationByReference	
Reservation	doReservationByTableNumber ()	IO method for getReservationByTableNumber	
Reservation	getReservationByNameAndContact (java.lang.String name, java.lang.String contact)	Get reservation by customer name and contact number	
Reservation	getReservationByReference (String referenceNumber)	Get reservation by reference number	