**Design Considerations:**

The objective of this project was to design a Restaurant Reservation and Point of Sale System (RPSS). An object oriented approach was taken towards designing this program. Our team took a modular approach and focused on the classes and objects we wanted to manipulate, how they relate to each other thereafter considering the logic we wanted to implement. The aim was to ensure the program is extensible and maintainable.

The RestaurantApp is the main class. This is the class that is run when the program is started. It is a boundary class that requires the user to decide the functions he wants the program to perform.

The following were four distinct groups of the remaining classes in our program:

1. Managing individuals:
2. Person, Member, Staff:   
   These three are basic entity classes that store information about individuals. Person is the base class containing attributes such as name, emailID etc. Member and Staff are derived classes extending from Person inheriting all of Peron’s attributes and methods, while also adding attributes exclusive to them (such as jobTitle).   
     
   The reason why Member and Staff were extended from Person is because they both share “is-a” relationship with Person. This eliminates the need for rewriting attributes and methods common to both Member and Staff, while also allowing for Member and Staff to have additional attributes depending on their need.
3. StaffList, MemberList:   
   These are the two control classes that perform manipulations on Staff and Member objects respectively such as create new staff/member, generate IDs, sort by IDs, print etc. The calls from the RestaurantApp class to manage Staff/Member pass through these.

StaffList and MemberList are ArrayList of Staff and Member objects respectively. Since the number of staf/ members in a restaurant is not pre-determined and can be changed from time to time, we have decided to use an ArrayList for dynamic memory allocation.

At the start of the program, on creation, these classes read in their respective list of objects from files and also store the final list of objects back into the files at the end of the program.

1. Managing orders placed by individuals:
2. Item, MenuItem, MenuSet, Order:   
   These are the entity classes within this category. MenuItem stores information about individual items on the menu such as name, description etc. MenuItem is the derived class of Item, that adds an attribute of integer datatype-category. Category is divided into Starter (1), Main (2), Dessert (3) and Drink (4).

The program allows user to create combinations of upto 10 items into a MenuSet by recording their unique itemIDs. MenuSet manages itself by providing methods to add/delete/update/check the existence of MenuItems within the set. These MenuSets could possibly be sold at cheaper prices (for instance Combos in restaurants).

Order class stores details about the order such as orderID, staffID of the staff creating the order, date, table assigned to the order, size etc.

1. Menu, PromoMenu, OrderList:   
   The control classes of this category, they manage MenuItems, MenuSet and Order classes respectively.

Menu stores ArrayList of MenuItems. The MenuItems in the ArrayList are read from a file and overwritten back into the file at the end of running the program, such that changes are updated. When the user wants to make changes to the restaurant’s menu such as add, update, remove items, then the RestaurantApp does this via the Menu class (via functions like createMenuItem). Similiarly PromoMenu is an ArrayList of MenuSets. It has the ability to create/update/remove/print MenuSets and is called by the Main class.

OrderList, in addition to creating, updating orders etc., is also used by RestaurantApp to calculate the net cost of an order and print the order receipt/invoice. While doing so, it checks if the customer is a member and provides for a 10% discount accordingly.

1. Managing Reservations:
2. Table, Reservation:   
   These are entity classes. Table Class contains the details about individual tables-tableID and capacity. Also, it has a boolean attribute isOccupied to display the status of the table. Reservation stores the details of reservations made by users such as start time, number of people, table ID, name, phone number etc.
3. TableList, ReservationList:

TableList is an array of ten tables that manages Table objects. It is used to change the status of tables when they get occupied or vacated and to check availability of tables. Interestingly, it can also be used to find the most suitable table for a group based on the difference between the capacity of the unoccupied tables and the number of people in the group.

ReservationList manages Reservation objects by creating, updating, checking for and printing reservations.

1. Display Report:
2. RevenueReport:   
   This is a boundary class that takes in input from the user to set the desired period and then displays the orders stored in the OrderList for dates that fall within the period.

**SOLID Design Principles:**

In order to ensure reusability, extensibility and maintainability and of the code, we have tried to incorporate the SOLID design principles in our code. The objective was to minimize the impact of change, to achieve low coupling and high cohesion. The principles that were incorporated in our program are as follows:

1. Single Responsibility Principle: SRP implies ensuring high cohesion in the program such that classes assume single responsibilities. All our entity classes such as Staff, Member, Table, Order etc. perform the single responsibility of storing information about a particular object type. Also the control classes such as Menu, MemberList, TableList, OrderList etc. are responsible for handling methods related to only one kind of object.
2. Open-Closed Principle: OCP requires a module to be open for extension but closed for modification. We have tried to implement this by means Person class and its subclasses—Member and Staff. Changes or modifications can be made to Member and Staff based on their individual needs, without requiring the Person class to change.
3. Don’t Repeat Yourself: In our code, every piece of knowledge such as Member, Staff, MenuItem etc is stored as a unique object such that any references to these within other classes point to the same object. This reduces the possibility of logical contradictions within the system.

In addition to this, principles of encapsulation and abstraction are illustrated in our program. Accessibility of attributes is private in most classes and internal representation of an object is hidden from outside view.

**Assumptions:**

1. It is assumed that there is only one type of Member who is provided with a 10% discount.
2. An individual MenuSet cannot contain more than ten items.
3. GST is assumed to be a universal rate of 7%.
4. Reservation cannot be updated. If there is a need to update reservation, a new reservation object is created.
5. Reservations are removed 30 minutes post the start time.
6. Existence of membership can only be checked by memberID, and not name or phone number.