1. Project Overview

The Restaurant Management System is designed to automate and streamline the workflow within a restaurant, covering essential operations such as menu management, order handling, billing, and customer and staff management. This system enables staff to manage customer orders efficiently, keep track of menu items, calculate bills, and handle customer memberships.

2. Functional Requirements

2.1 Menu Management

- Add, update, and delete menu items.
- Support for menu attributes including name, price, category, description, preparation time, and availability.
- Apply discounts to items that are on sale.

2.2 Customer Management

- Add and manage customer records, including membership status.
- Apply special discounts or benefits for members.

2.3 Order Processing

- Allow customers or waitstaff to create and manage orders.
- Add and remove items within an order.
- Calculate the total order price.

2.4 Billing

- Generate bills for completed orders, applying any applicable discounts.
- Display and save bills to a file for record-keeping.

2.5 Staff Management

- Manage waiter and cashier details, enabling assignment to specific orders or tables.

2.6 Persistence

- Save and load data for menu items, customers, waiters, and bills using file input/output.

3. Non-Functional Requirements

- Usability: User-friendly console interface with clear prompts.

- Reliability: Ensure data consistency across orders, billing, and menu updates.
- Scalability: Easily adaptable to different restaurant sizes and menu complexity.

4. System Architecture

The system follows an object-oriented approach, with classes representing core entities and their interactions. Linked lists are used to manage collections of objects, allowing dynamic memory allocation for menu items, customers, and staff.

5. Class Structure

5.1 Classes Overview

- Menu
  - Represents a menu item with attributes such as name, price, availability, category, description, preparation time, sale status, and discount.
  - Linked-list pointers (next, prev) for doubly linked list implementation.
- MenuLL
  - Manages a list of Menu items, providing methods to add, remove, update, and retrieve menu items.
  - Save in file and read from file for default setting.
- Menu Stack
  - A data structure followed by a First-In-Last-Out order keeps track of deleted menu objects and ensures Undo functionality.
- Person
  - Super class of Customer class and Waiter class, with attributes inherited by both sub-classes.
- Customer
  - Represents a customer with attributes like name, contact information, and membership status.
  - Linked-list pointers for managing a linked list.
- CustomerLL

    - Stores a linked list of customers and includes methods to add, update, search, and remove customer records.
    - Save in file and read from file for default setting.
- Customer Stack
    - A data structure followed by a First-In-Last-Out order keeps track of deleted Customer items and ensures Undo functionality.
- Waiter
    - Represents a waiter with attributes like name, contact information, and waiter id.
    - Linked-list pointers for managing a linked list.
- WaiterLL
    - Stores a linked list of waiters and includes methods to add, update, search, and remove waiter records.
    - Save in file and read from file for default setting.
- Waiter Stack
    - A data structure followed by a First-In-Last-Out order keeps track of deleted Waiter Class items and ensures Undo functionality.
- Bill
    - Calculates the final bill for an order, including discounts.
    - Keep track of ordered items for each table (Restaurant object).
    - Waiter and Cashier
    - Represents staff members, with Waiter or Cashier-specific attributes.
    - Linked-list pointers to manage a list of staff members.
- Restaurant
    - Represents each table in the restaurant with attributes like the corresponding waiter, Bill, and unique table number.
    - Linked-list pointers for managing a linked list
- RestaurantLL
    - Stores a linked list of restaurants and includes methods to add, update, search, and remove customer records.
    - Save in file and read from file for default setting.
- Restaurant Stack

- A data structure followed by a First-In-Last-Out order keeps track of deleted Restaurant Class items and ensures Undo functionality

6. Methods and Key Operations

6.1 Menu Management (MenuLL Class)

- addMenu(Menu menu): Adds a new item to the menu.
- updateMenu(int id, Menu newDetails): Updates menu item details by ID.
- removeMenu(int id): Removes a menu item by ID.

6.2 Customer Management and Waiter Management (CustomerLL , WaiterLL Class)

- addCustomerEnd(Customer customer): Adds a new customer record.
- deleteWaiterById(int id): Removes a customer record.
- addWaiterEnd(Waiter waiter): Adds a new waiter record.
- deleteWaiterById(int id): Removes a waiter record.

6.3 Table Management (RestaurantLL Class)

- addRestaurantEnd(Restaurant restaurant): Adds a restaurant item to the order.
- deleteSearch(int tableNumber): Removes an item from the order by menu ID.

6.4 Billing (MenuLL Class, Bill Class)

- getSubtotal(): Calculate the subtotal of a bill

7. File Handling and Persistence

Data for menu items, customers, and bills will be stored using object serialization in Java. Each list class (e.g., MenuLL, CustomerLL) includes methods to save and load its items to/from a file, ensuring that data persists across application sessions.

8. User Interface Design

The system will be console-based, providing text menus for navigation. Each menu will offer options relevant to the current module, such as adding or updating menu items, managing orders, and generating bills.

Main Menu: Provides options to access Menu Management, Customer Management, Ordering, and Billing.

Submenus: Each feature (e.g., adding a menu item, creating an order) will have its own submenu with options for the relevant operations.

9. Error Handling

- Null Checks: Validate object references (e.g., checking for null when searching for an item by ID).
- Exception Handling: Use custom exceptions (e.g., IOException) for specific cases of io exception.
- Input Validation: Validate user input (e.g., correct data types for prices, valid IDs).

10. Testing Strategy

10.1 Unit Testing

Test individual classes and methods, such as adding/removing menu items, generating bills, and customer updates.

10.2 Integration Testing

Test interactions between classes, especially linked list structures and how they handle CRUD operations.

10.3 User Acceptance Testing (UAT)

Test the full flow of the application to ensure it meets functional requirements and is intuitive to use.

11. Future Enhancements

- Database Integration: Replace file storage with a relational database for larger data sets.
- Graphical User Interface (GUI): Develop a GUI for better user experience.
- Advanced Membership and Loyalty Programs: Include more complex loyalty point systems and reward tracking.