# Restaurant Order Management System

## Project Overview

The Restaurant Order Management System is a comprehensive application designed to streamline the management of orders in a restaurant setting. The system utilizes the Model-View-Controller (MVC) architectural pattern to separate concerns, making it easier to manage, maintain, and extend. This project provides a robust solution for handling customer orders, managing payments, and tracking orders efficiently. It integrates various components such as menu management, order processing, customer management, and payment simulation.

# Features

1. **Restaurant Setup:**
    - The system allows for the creation of a restaurant with a name and menu.
    - Menu items can be added or removed, categorizing them as appetizers, main courses, desserts, etc.
2. **Menu Management:**
    - Add/Remove Menu Items: Users can add or remove menu items from the restaurant's menu.
    - View Menu: Displays available items with details like name, price, and category.
3. **Order Management:**
    - Create Orders: Customers can place orders by selecting items from the menu.
    - Order Items: Each order consists of multiple items selected from the menu.

- Order Validation: The system validates orders to ensure they meet the business rules (e.g., must contain at least one item).
- Order Processing: Orders are processed through a workflow that changes their status from PENDING to PREPARING and finally to COMPLETED.
- Order Management: Users can view, edit, or cancel orders.

4. **Payment Simulation:**
    - Payment Methods: Supports payment through credit card or cash.
    - Payment Processing: Simulates payment transactions based on the selected payment method.
    - Discounts: Allows the application of discounts to orders, reducing the total amount.
5. **Customer Management:**
    - Customer Details: Manages customer details including ID, name, and contact information.
    - Customer Orders: Tracks orders placed by customers.
6. **View Layer:**
    - Interactive UI: Presents restaurant information, menu, orders, and payment status in a user-friendly manner.
    - Real-time Updates: Reflects changes in orders and payments to the user immediately.

## Project Structure

**T**he project follows the MVC (Model-View-Controller) design pattern, ensuring a clear separation of concerns. Here's a breakdown of the project structure:

```
RestaurantManagementSystem/
│
├── src/
│   ├── controller/          # Controller classes
│   │   └── RestaurantController.java
│   │
│   ├── model/               # Model classes
│   │   ├── menu/            # Menu-related models
│   │   │   ├── Menu.java
│   │   │   └── MenuItem.java
│   │   │
│   │   ├── order/           # Order-related models
│   │   │   └── Order.java
│   │   │
│   │   ├── customer/        # Customer-related models
│   │   │   └── Customer.java
│   │   │
│   │   ├── employee/        # Employee-related models
│   │   │   ├── Employee.java
│   │   │   ├── Chef.java
│   │   │   └── Waiter.java
│   │   │
│   │   ├── payment/         # Payment-related models
│   │   │   ├── Payment.java
│   │   │   ├── PaymentProcessor.java
│   │   │   ├── CashPayment.java
│   │   │   └── CardPayment.java
```

```
│   │
│   └── view/              # View classes
│       └── RestaurantView.java
│
├── resources/             # Any static resources (e.g., images, etc.)
│   └── images/            # (if required)
│
├── lib/                   # External libraries (if any)
│
├── build/                 # Compiled files (for IDEs like IntelliJ, Eclipse)
│
├── README.md              # Project description
├── .gitignore             # Git ignore file
└── pom.xml / build.gradle    # Dependency management (for Maven or Gradle)
```

# MVC Architecture of the Project

1. **Model:**
    - **Responsibilities:**
        - Stores and manages application data.
        - Encapsulates business rules and logic.
        - Provides methods for data manipulation (e.g., adding/removing menu items, validating orders).
    - **Components:**
        - RestaurantModel: Represents the restaurant, including its menu and orders.
        - MenuModel: Manages the restaurant's menu items.
        - MenuItemModel: Represents individual menu items.

- OrderModel: Represents a customer's order.
- OrderItemModel: Represents an item within an order.
- CustomerModel: Represents customer details.
- OrderValidatorModel: Validates orders.
- OrderProcessorModel: Processes orders.
- PaymentModel: Handles payment transactions.
- DiscountModel: Applies discounts to orders.

2. **View:**
    - **Responsibilities:**
        - Displays data from the model in a user-friendly format.
        - Provides an interface for user interaction.
    - **Components:**
        - RestaurantView: Displays restaurant information, menu, orders, and payment status.
        - Methods include displaying restaurant details, menu items, customer details, and processing messages.
3. **Controller:**
    - **Responsibilities:**
        - Manages user input and coordinates the flow of data between the model and view.
        - Directs actions based on user input (e.g., processing orders, handling payments).
    - **Components:**
        - OrderProcessorModel: Controls order processing and payment transactions.
        - Interacts with the model to validate and update orders and with the view to reflect changes.
        - PaymentModel: Manages payment processing.
        - Coordinates with the model to simulate payments.

## Benefits of the MVC Architecture in this Project:

- Separation of Concerns: By keeping the model, view, and controller separate, the architecture enhances modularity and makes the codebase easier to manage and extend.
- Scalability: New features can be added without disrupting the existing codebase.
- Reusability: Each component can be reused independently in different contexts or applications.
- Simplified Testing: The clear separation allows for unit testing each component independently, making the system more robust.