Swinburne University of Technology

Restaurant Information System – Object Design

SWE30003

By:
Marco Giacoppo 104071453
Corey Santarossa 103389809
Thai Duc Nguyen 103806719
Rafia Sanjida Chowdhury 103177473

Table of Contents

Executive Summary

The Relaxing Koala, a popular café/restaurant located on Glenferrie Road, is poised for a significant operational shift. Following the acquisition of an adjoining property, the venue is expanding its capacity from 50 to 150 seats. This expansion necessitates a transition from a largely manual, low-tech operation to a more sophisticated, automated restaurant information system. The primary objective is to enhance efficiency in reservations, order management, kitchen operations, invoicing, and payments, while also providing a platform for future integration of online ordering and delivery services

Introduction

This document outlines the object design for the Relaxing Koala's new restaurant information system. It serves as a comprehensive guide for developers, managers, and stakeholders, facilitating clear communication regarding the system's architecture and operational dynamics. This design is crucial for ensuring efficient transition from manual to automated processes in restaurant management

Assumptions

- All user interactions are conducted through the system interface
- The restaurant has a digital menu that can be updated in real-time
- Payment processing is secure and compliant with financial regulations

Evidence of Problem Analysis

The initial design of the Relaxing Koala Restaurant Information System was guided by a thorough Software Requirements Specification (SRS), which included a detailed requirements analysis. This applies to these key functionalities:

Key Functionalities Identified:

1. Reservation Management:

   Record Reservation Details: Ability to capture and store specifics of customer reservations including date, time, number of guests, and special requests

Manage Reservation Statuses: Updating and tracking the status of each reservation (e.g., pending, confirmed, canceled)

2. Order Processing:

    Record and Update Order Details: Capture each order's content, including specific menu items and quantities

    Manage Order Lifecycle: Track orders from placement through to preparation and service, ensuring timely updates to both staff and customers

3. Payment Processing:

    Record Payment Transactions: Log details of each transaction, including payment method and status

    Handle Multiple Payment Methods: Support a variety of payment options such as cash, credit cards, and mobile payments

4. Customer Management:

    Store Customer Data: Maintain a database of customer information for both regular and one-time guests

    Track Customer Preferences: Record and analyze customer preferences to enhance personalized service and marketing efforts

5. Kitchen Workflow Integration:

    Communicate Order Details to Kitchen: Ensure that order details are accurately and promptly relayed to the kitchen staff

    Update Order Status: Inform front-of-house staff about the status of meal preparation

6. Reporting and Invoicing:

    Generate Real-Time Reports: Provide reports on sales, customer visits, and popular menu items

    Create and Manage Invoices: Automatically generate invoices upon order completion, ensuring accuracy and efficiency

Simplifications :

In developing the initial object-oriented design for the Relaxing Koala Restaurant Information System, several simplifications have been made to manage complexity, ensure clarity in the development process, and align the system design with the practical operational needs of a mid-sized restaurant. These simplifications are grounded in realistic assumptions about the restaurant's operations and potential technological constraints:

- We're prioritizing essential modules: Reservations, Orders, Kitchen Management, Payments, and Invoicing. This focus allows for a concentrated development effort on critical functionalities that directly enhance operational efficiency and customer service

- Utilization of data-holder classes simplifies data handling by separating data storage from business logic, facilitating clearer system structure and easier maintenance
- The system will start with a fundamental user interface, supporting essential interactions for bookings, orders, and payments, deferring more complex UI features to later phases
- Integration with existing POS systems and databases minimizes development time and leverages proven technologies, simplifying the technological footprint
- The system will initially support standard payment methods (cash, credit/debit cards), focusing on the most commonly used options to streamline the payment process
- Assuming a reliable network within the restaurant simplifies the initial network error handling requirements, allowing the team to focus more on system functionality

Overview of Candidate Classes

In the restaurant information system design, the UML class diagram in **Figure 1** below features a set of classes that represent the key entities and their relationship within the system. These classes are divided into two categories: operational classes that manage the logic and behavior of the system, and data-holder classes that serve as containers for data without behavior.

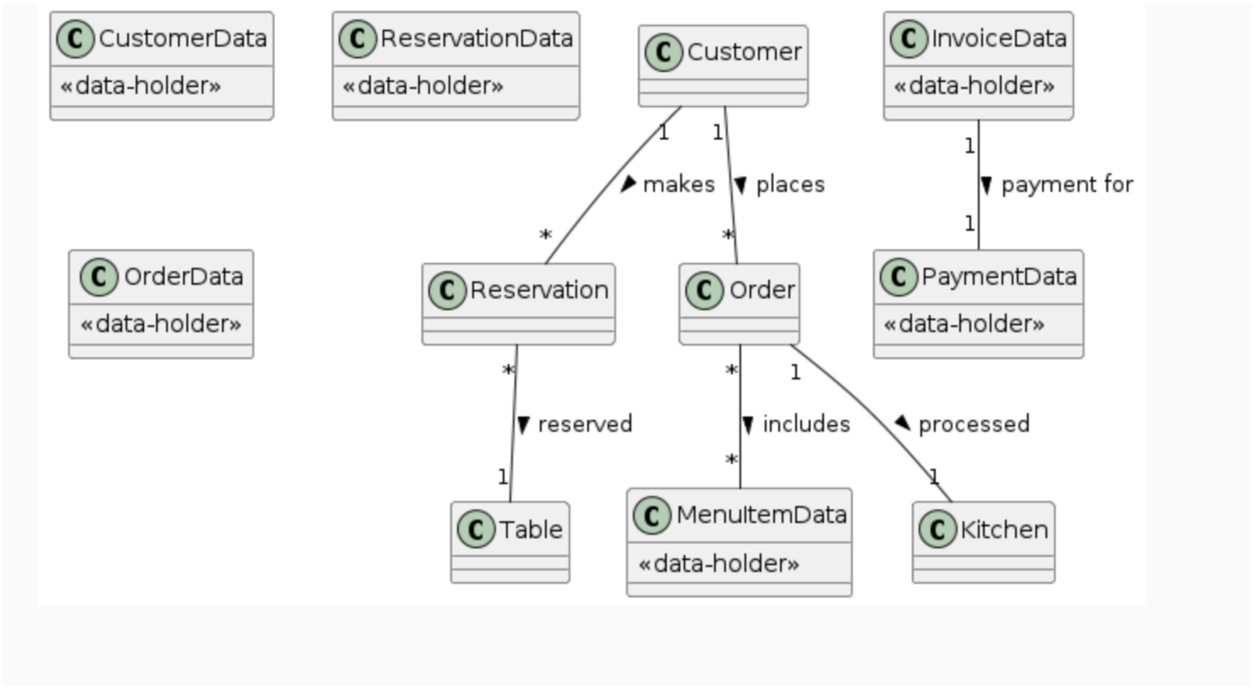

Figure 1:UML Class Diagram of the Restaurant Information System

Operational Classes:

1. **Customer:** Represents the end-users of the system. This class is responsible for activities that directly involve customer interaction, such as making reservations and placing orders. The relationship between Customer and Reservation, as well as Customer and Order, is essential as it allows the system to track which reservations and orders belong to which customers.
2. **Reservation:** Manages all reservation-related activities. This class works closely with the Customer and Table classes to book tables and manage the status of reservations, ensuring that customers can reserve tables and that the restaurant can effectively manage its seating capacity.
3. **Order:** Handles the processing of customers' orders. It collaborates with the Kitchen to ensure that orders are prepared and served efficiently. This class is a conduit between the customer's dining experience and the kitchen's operation.
4. **Kitchen:** Represents the back-end processes involved in order preparation. To guarantee that meals are prepared in accordance with received orders and that the status of orders is updated during the cooking process, this class must efficiently communicate with the Order class.
5. **Table:** Represents the actual restaurant tables. In order to help the restaurant efficiently manage its dining space, this class communicates with the Reservation class to keep track of which tables are available or reserved.

Data-holder Classes:

1. **CustomerData:** Contains all of the pertinent information about a customer, including their name, contact information, and other pertinent details. The Customer class uses it to handle data pertaining to customers.
2. **ReservationData:** Contains details about reservations, including the date, time, and number of guests. This data is used by the Reservation class to keep track of reservation specifics.
3. **OrderData:** Stores information regarding customer orders, such as the items ordered, quantities, and special requests. The order class uses this data to process and manage orders.
4. **MenuItemData:** Holds information about each menu item, including descriptions, prices, and availability. It provides the necessary data for the Order class to include items in customer orders.
5. **InvoiceData:** Contains details needed for billing, such as itemized costs and total amounts. It is utilized by the Invoice operational class to generate accurate bills for customers.
6. **PaymentData:** Stores transaction data, including payment amounts, methods, and statuses. The Payment class uses this data to process and record payments.

The system shown in the diagram is intended to manage restaurant operations on a daily basis, including customer interactions and kitchen workflow. The separation of data structures from the operations that manipulate them, along with the clear definition of roles and responsibilities for each class, are the fundamental components of responsibility-driven design, which is reflected in the separation of operational and data-holder classes.

The data-holder classes act as the information repositories that the operational classes need to carry out their tasks, and the relationships depicted in the UML diagram represent the interactions required to carry out these responsibilities. The system is made with a focus on modularity, distinct concern separations, and scalability and maintainability in mind.

CRC Cards for Candidate Classes

Component: Customer	
SuperClasses: n/a	**SubClasses:** n/a
Responsibilities	**Collaborators**
Provide personal details (name, contact information)	n/a

Make reservations	Reservation
Place orders	Order
Receive updates on order status	Order

Explanation:

The Customer class represents the end-user of the restaurant information system. It holds essential details about the customers, including their contact information which is crucial for managing reservations and orders. The class allows customers to interact with the system, such as making and tracking their reservations and orders, ensuring a personalized and efficient customer service experience.

Component: Reservation	
SuperClasses: n/a	**SubClasses:** n/a
Responsibilities	**Collaborators**
Record reservation details (date, time, number of guests)	Customer, Table
Manage reservation statuses (booked, cancelled, confirmed)	Table
Check and update table availability	Table
Handle special requests	Customer

Explanation:

The Reservation class manages all aspects of booking a table at the restaurant. It is designed to hold the reservation's specifics and keep track of the status of each booking. Collaborating closely with the Customer class for inputs and the Table class for availability checks, it ensures that reservations are managed effectively, reducing the chance of overbooking, and improving the utilization of the restaurant's seating capacity.

Component: Order	
SuperClasses: n/a	**SubClasses:** n/a
Responsibilities	**Collaborators**
Track order items	MenuItem
Calculate and update total cost	MenuItem, Invoice

Manage order lifecycle (open, in preparation, served)	Kitchen, Customer
Communicate with Kitchen to update on order status	Kitchen

Explanation:

The Order class is central to the restaurant's operations, tracking each customer's selection of menu items. It's responsible for calculating the total cost, incorporating any additions or deletions to the order, and managing the lifecycle of the order as it progresses from being placed to being served. The Order class collaborates closely with Customer to reflect their choices, MenuItem to detail the items ordered, Invoice to generate a bill for payment, and Kitchen to ensure that the meal is prepared according to the order specifications.

Component: MenuItem	
SuperClasses: n/a	**SubClasses:** n/a
Responsibilities	**Collaborators**
Maintain item details (name, description, price)	n/a
Update inventory status	n/a
Provide information for order customizations	Order

Explanation:

The MenuItem class acts as a digital representation of the restaurant's offerings, detailing each dish or drink, its price, and description. It's a vital part of the order process, providing the necessary information to both the customer placing the order and the kitchen preparing it. The class also includes functionality to update the availability of items, which helps in managing the inventory and informing customers about out-of-stock items.

Component: Invoice	
SuperClasses: n/a	**SubClasses:** n/a
Responsibilities	**Collaborators**
Generate itemized bill from orders	Order
Process payments	Payment
Record transaction details	Payment

Provide digital or printed receipts	Customer

Explanation:

The Invoice class encapsulates financial transactions within the restaurant. It generates a detailed bill based on the customer's order and handles the payment process, ensuring accurate and secure transactions. The class is responsible for issuing receipts and maintaining a record of all financial activities, which is crucial for accounting and customer service.

Component: Payment	
SuperClasses: n/a	**SubClasses:** n/a
Responsibilities	**Collaborators**
Handle payment processing (credit card, cash, online)	Invoice
Validate payment methods and details	n/a
Securely manage and store transaction data	n/a

Explanation:

The Payment class is responsible for all aspects of transaction processing within the restaurant information system. It supports various payment methods, ensuring flexibility and convenience for customers. The class collaborates with Invoice to finalize transactions and is designed to safeguard sensitive financial data, adhering to the highest security standards.

Component: Kitchen	
SuperClasses: n/a	**SubClasses:** n/a
Responsibilities	**Collaborators**
Prepare dishes as per order requests	Order
Manage kitchen staff and equipment	n/a
Communication with front of house regarding order status	n/a

Explanation:

The Kitchen class is at the heart of the restaurant's food preparation operations. It manages the flow of orders from the dining area to the kitchen staff, ensuring that meals are prepared correctly and efficiently. The class facilitates communication between the kitchen and the service staff, enabling timely updates about order statuses and contributing to a smooth dining experience.

Component: Table	
SuperClasses: n/a	**SubClasses:** n/a
Responsibilities	**Collaborators**
Track occupancy status	Reservation
Hold information about location and capacity	n/a
Be reserved or released based on customer arrivals/departures	Reservation

Explanation:

The Table class is an abstraction of the physical dining space within the restaurant. It keeps track of each table's occupancy status and features such as capacity and location. By collaborating with the Reservation class, it helps to efficiently manage the restaurant's seating arrangements, thereby optimizing space utilization and customer flow.

Quality of Design Solution

Design Patterns

Observer Pattern

The Observer pattern is used within the design to enable real-time updates across various classes. For example, the Order class alerts the Kitchen and Customer classes when an order is made. This pattern avoids tight coupling between classes, allowing for flexible and modular updates which enhance the systems responsiveness and adaptability.

Factory Pattern

The software design uses the Factory pattern during the bootstrap/initialisation process. For example, Data classes such as CustomerData and ReservationData are used when creating the operational classes Customer and Reservation. This pattern controls object creation through a factory interface, simplifying object creation. Implementing the Factory pattern ensures standardised object creation which is critical for system integrity and scalability.

Strategy Pattern

The Relaxing Koala design uses the Strategy pattern for handling payment methods as they can vary between cash, credit card, and online payments. This pattern allows for different payment algorithms to exist within the Payment class, enabling it to switch between algorithms based on the payment method used. This pattern allows for simplified payment handling, allowing for flexibility and scalability across all payment methods.

Design Heuristics

Heuristic 2.8: A class should capture one and only one key abstraction.
Explanation: Each class in the design has a single main abstraction. For example, the Order class focuses on handling orders, and the Customer class focuses on customer information and interactions.

Heuristic 2.9: Keep related data and behaviour in one place.
Explanation: Classes such as Reservation, Order, and Payment encapsulate both the methods and data needed for that class to function successfully.

Heuristic 3.1: Distribute system intelligence horizontally as uniformly as possible, that is, top-level classes of the system should share their work uniformly.
Explanation: No individual class has excessive responsibilities. Complicated tasks are performed from the collaboration of multiple classes.

Heuristic 3.2: Do not create God classes/objects in your system.
Explanation: There is no God class in this design. Each class has well defined roles and responsibilities, where no class has Master privileges.

Illustration of Bootstrap Process

The following bootstrap process demonstrates how each class and component of the Relaxing Koala application is initialised.

1. As the application starts, the initialisation of CustomerData, ReservationData, OrderData, MenuItemData, InvoiceData, and PaymentData begin, allowing the system to prepare with all necessary data holders.
2. CustomerData is used to create and populate a new Customer instance.
3. The Customer instance communicates with the Reservation class to create a reservation. It uses ReservationData and verifies table availability through the Table class.
4. When a customer places an order, an Order instance is created, which collects menu item information from the MenuItemData data holder.
5. When new orders are placed, a Kitchen instance receives the order allowing the staff to prepare the dishes, with details provided by the Order class.
6. After preparation, the Invoice class is instantiated, which uses InvoiceData to create the appropriate bill for the customer.

7. The Customer completes the transaction using the Payment class, which processes the payment through PaymentData.
8. This process ends with all classes operational, marking the end of the Relaxing Koala bootstrap process.

Basic Verification of Given Solution

To validate the design of the proposed restaurant information system, it is essential to rigorously evaluate it through a series of detailed, non-trivial use scenarios. These scenarios are designed to simulate typical operations within a restaurant environment, providing a comprehensive view of how the system manages and facilitates routine interactions among the various classes. By illustrating these interactions with UML sequence diagrams, we can effectively depict the chronological flow of messages between objects, thereby highlighting the system's capabilities and confirming its alignment with operational requirements. This method not only clarifies the dynamic interactions within the system but also ensures that the design adheres to the specified functional and user experience standards.

Scenario 1: Making a Reservation

Process:

- A customer accesses the system to make a reservation.
- The Customer class captures the customer's details and interacts with the Reservation class.
- The Reservation class checks table availability via the Table class.
- Once confirmed, the Reservation class records the reservation details as shown in **Figure 2** below.

Sequence Diagram:

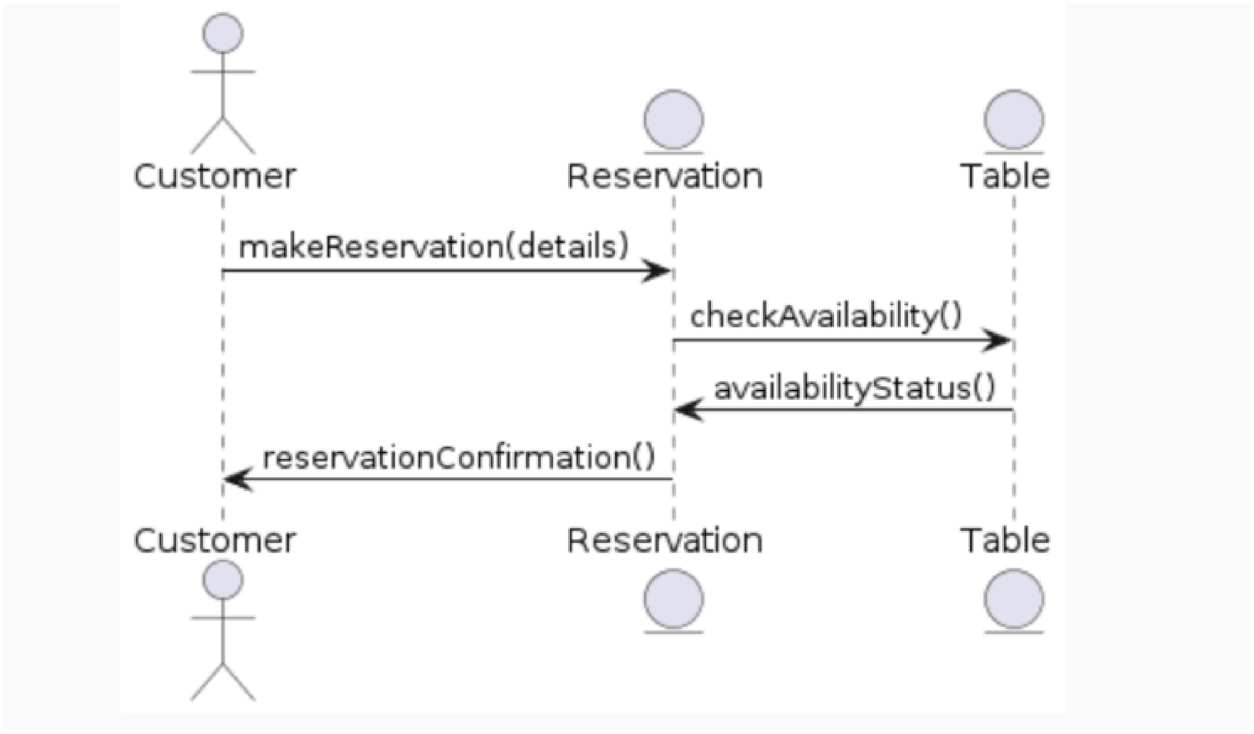

Figure 2: UML Sequence Diagram for Making a Reservation

Scenario 2: Placing an Order

Process:

- A customer decides to order food.
- The Order class is instantiated, pulling menu item details from MenuItemData.
- The customer selects items, and the Order updates the total cost.
- The Order informs the Kitchen to prepare the dish as shown in **Figure 3** below.

Sequence Diagram:

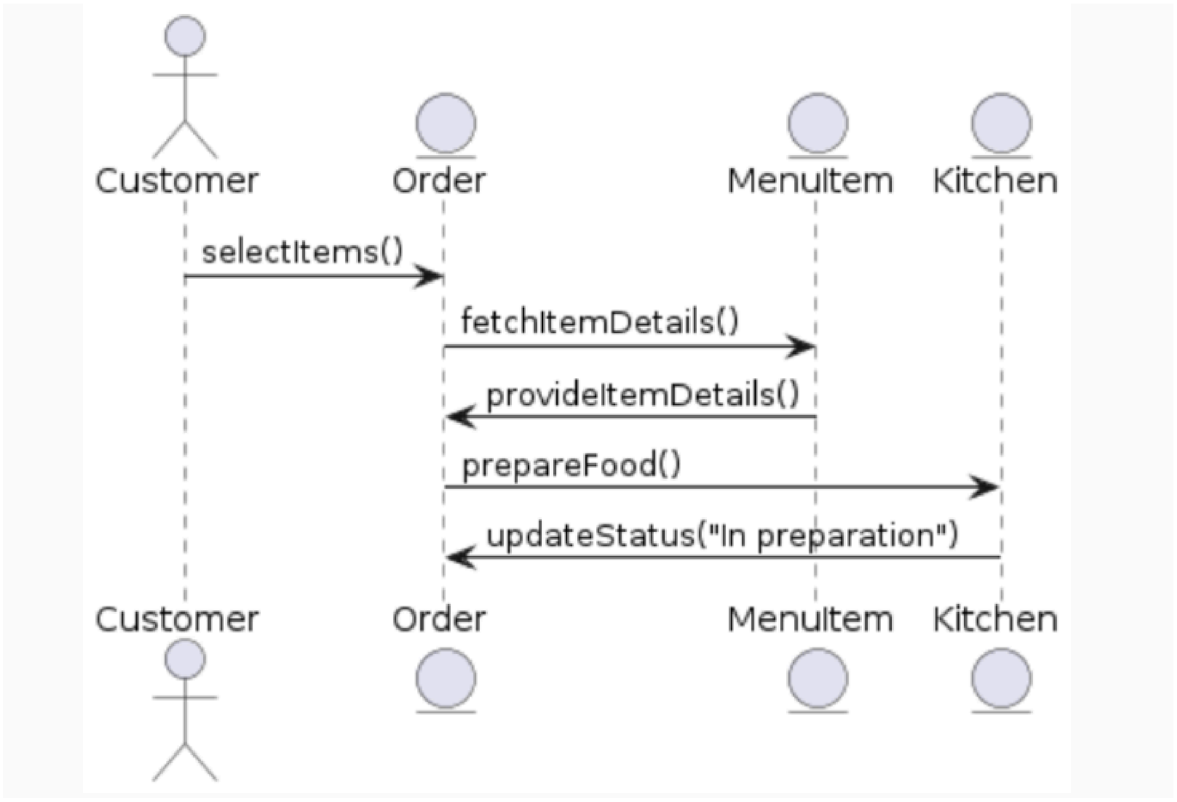

Figure 3: UML Sequence Diagram for Placing an Order

Scenario 3: Payment Processing

Process:

- After dining, the customer requests the bill.
- The Invoice class generates an itemized bill using details from the Order.
- The customer makes a payment through the Payment class, which processes the transaction as shown in **Figure 4** below.

Sequence Diagram:

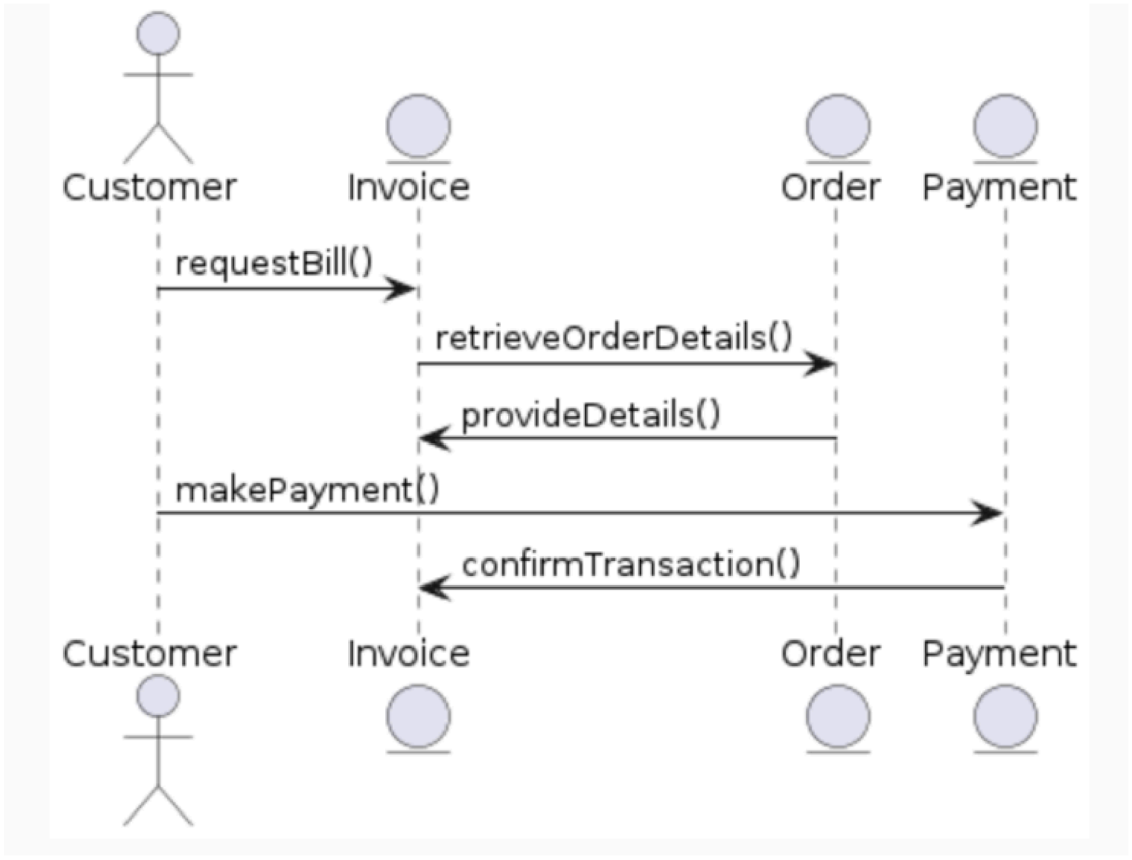

Figure 4: UML Sequence Diagram for Payment Processing

Scenario 4: Handling Special Requests

Process:

- During reservation or ordering, the customer makes special requests (e.g., dietary restrictions).
- The Reservation or Order class records these details and ensures they are communicated to the Kitchen or relevant staff as shown in **Figure 5** below.

Sequence Diagram:

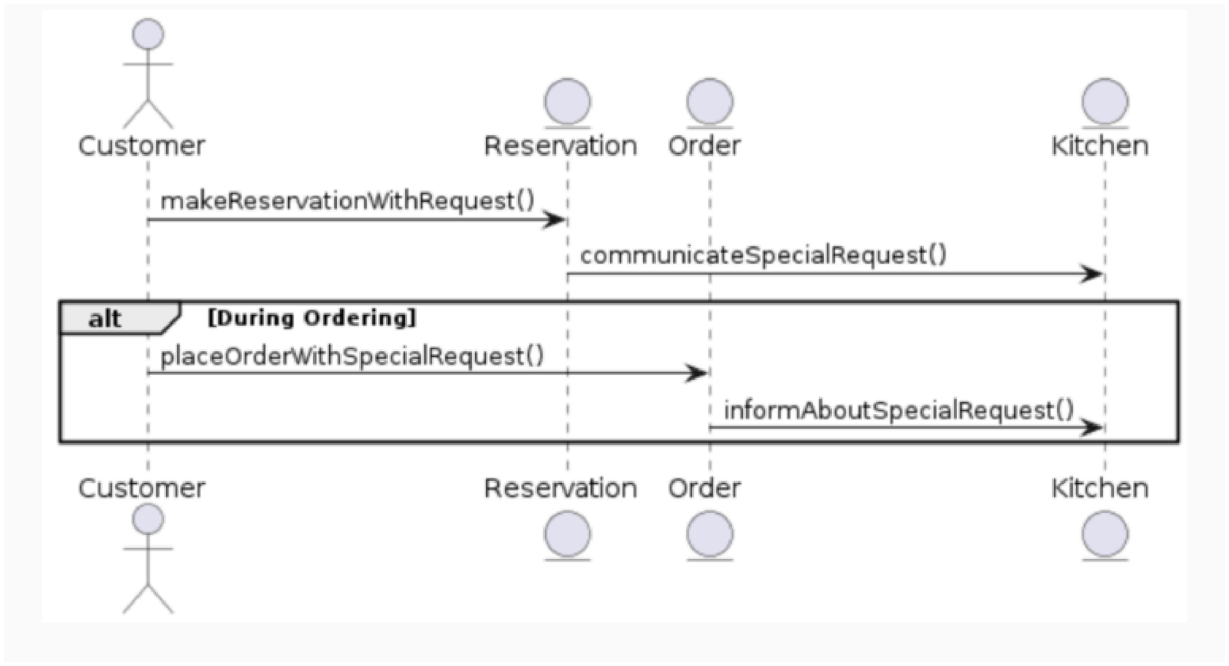
Customer
Reservation
Order
Kitchen
makeReservationWithRequest()
communicateSpecialRequest()
alt
[During Ordering]
placeOrderWithSpecialRequest()
informAboutSpecialRequest()
Customer
Reservation
Order
Kitchen

Figure 5: UML Sequence Diagram for Handling Special Requests