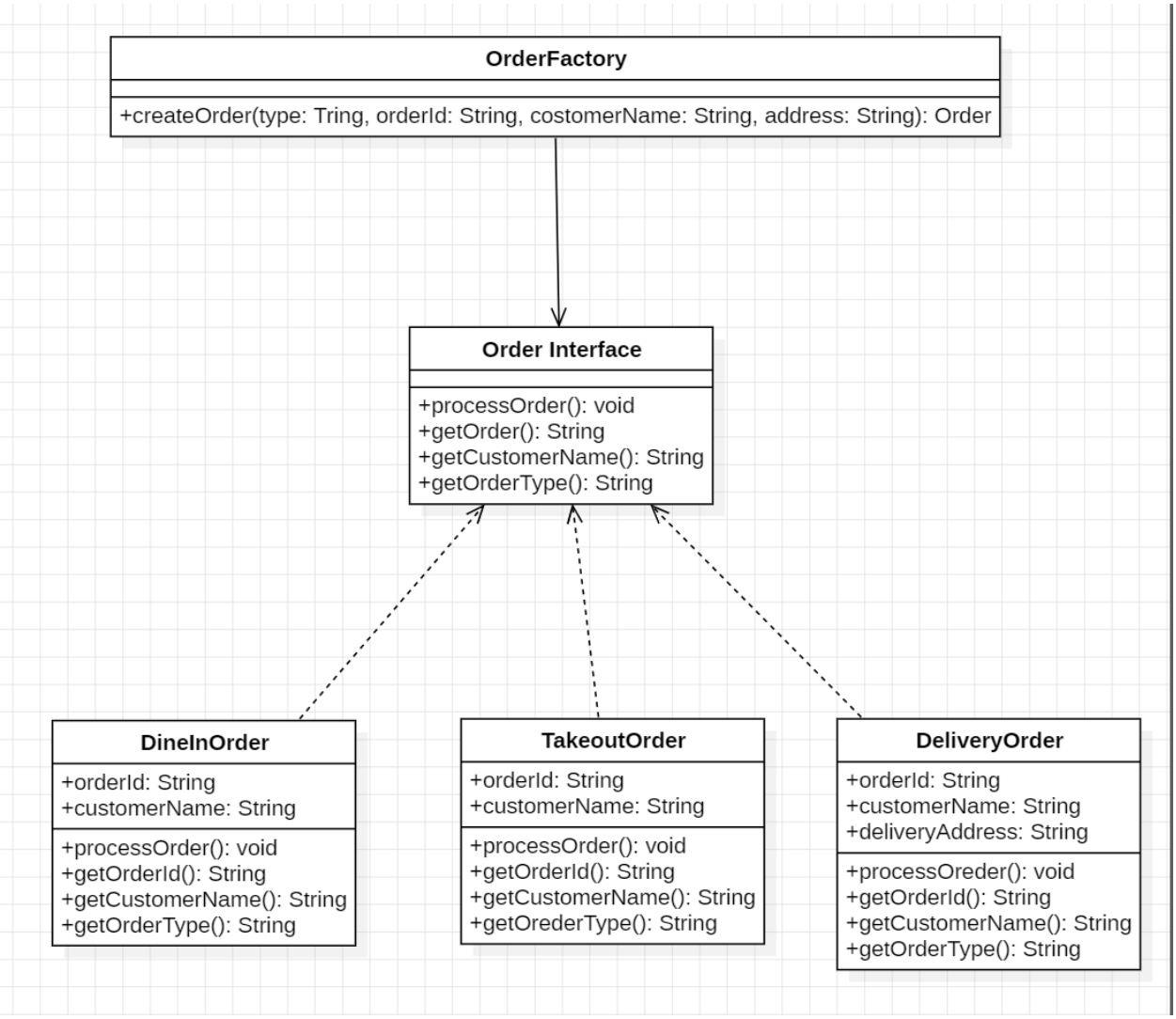
Joldi Xure

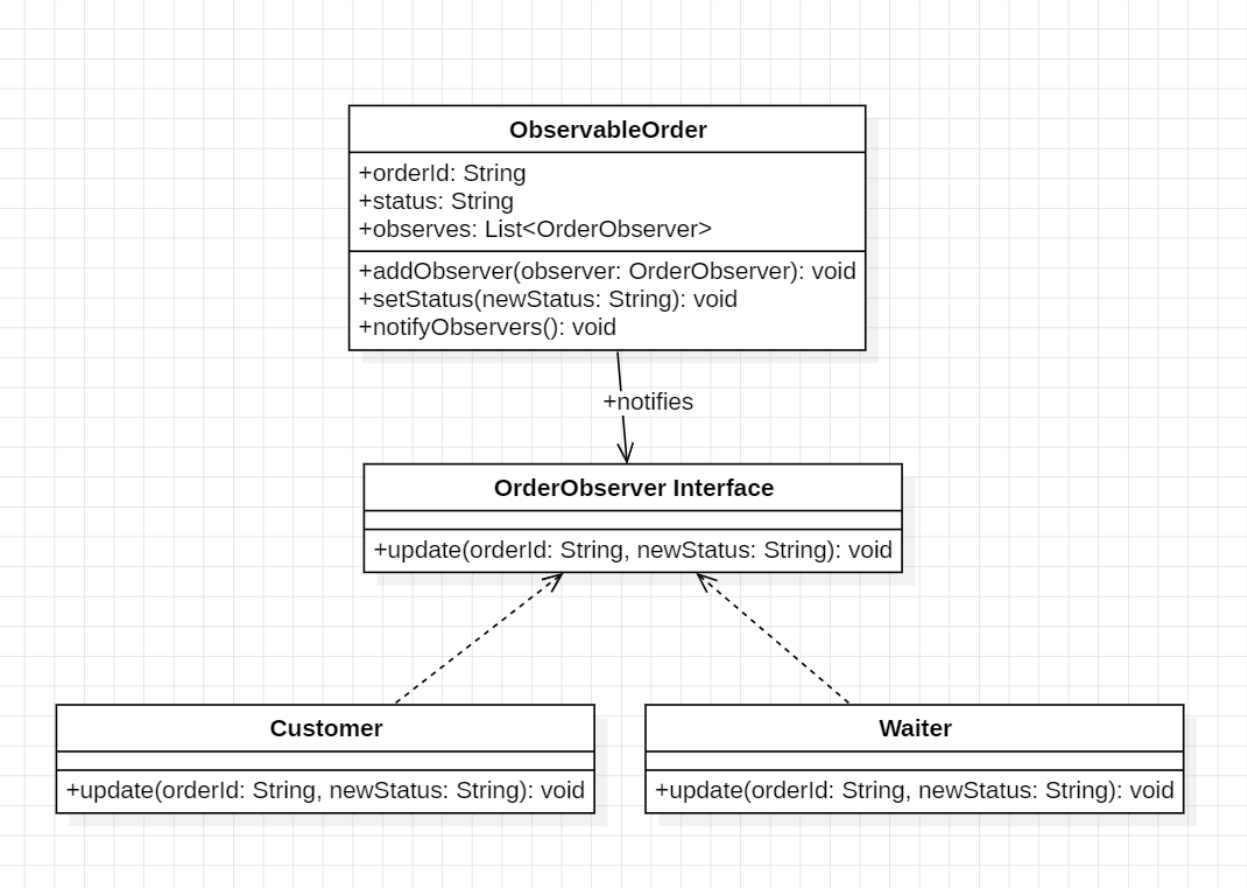
Factory Method Pattern

The Factory Method pattern is employed to abstract the process of object creation, allowing the system to instantiate specific order types (such as DineInOrder, TakeoutOrder, or DeliveryOrder) without coupling the core application logic to concrete class implementations. In the context of a restaurant management system (RMS), this approach promotes extensibility and maintainability by centralizing the instantiation logic within a dedicated factory class. This ensures that new order types can be integrated with minimal impact on existing code, in line with the Open/Closed Principle of object-oriented design. It also supports consistent instantiation behavior and reduces duplication, making the system more scalable and easier to evolve over time.



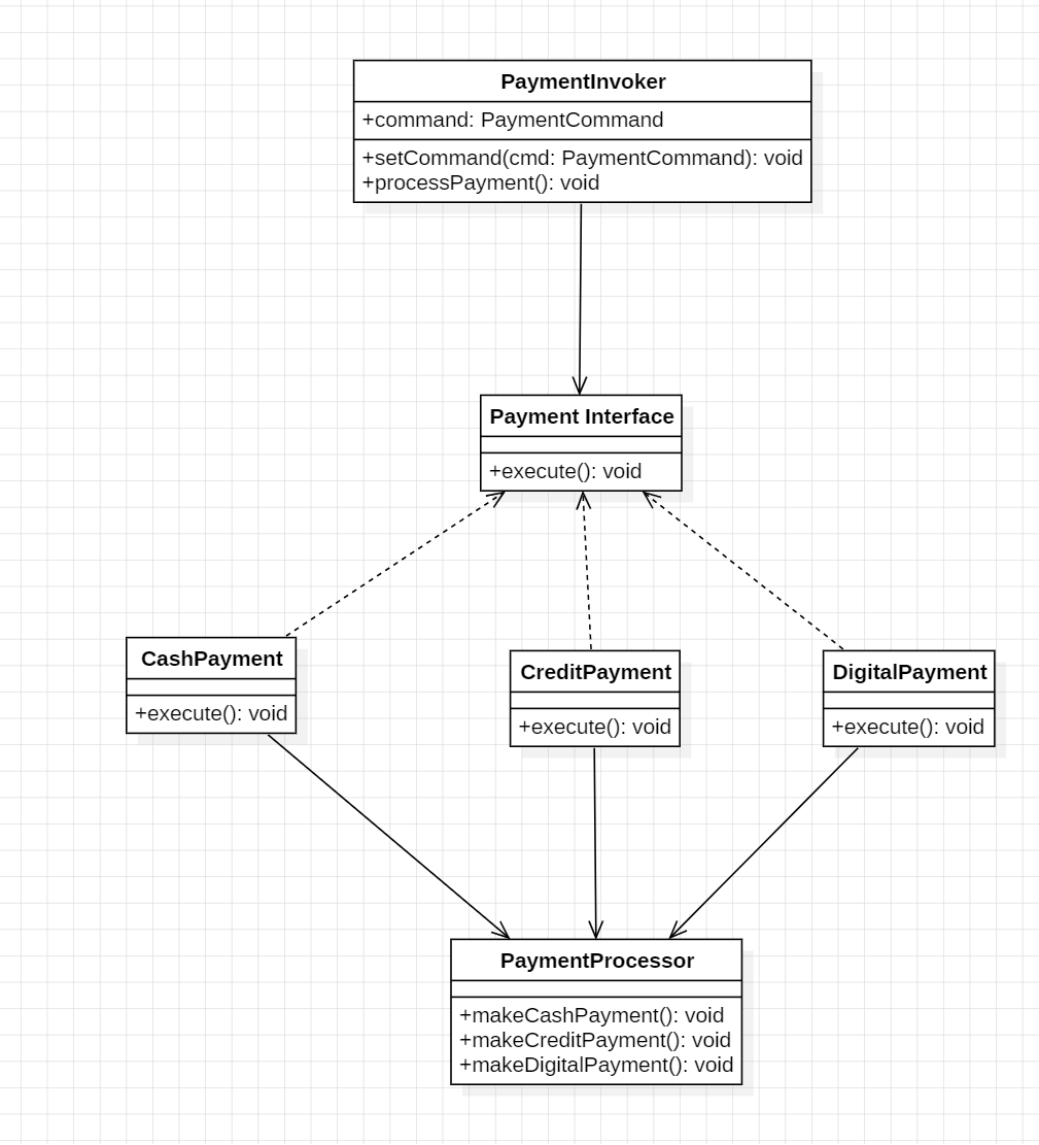
Observer Pattern

The Observer pattern is implemented to establish a one-to-many dependency between the order state and its listeners, such that when the status of an order changes (e.g., from "Preparing" to "Ready"), all subscribed entities—such as customers, waitstaff dashboards, or display systems—are automatically notified. This design is critical in a restaurant management system where timely communication and real-time updates are essential for operational efficiency. By decoupling the order management logic from the various user interfaces, the pattern enhances system modularity and scalability, enabling seamless integration of new notification channels or devices without modifying the core logic.



Command Pattern

The Command pattern is utilized to encapsulate payment-related actions (e.g., cash, credit card, or digital payment) as standalone command objects that implement a unified interface. This abstraction allows the restaurant management system to decouple the initiation of a payment operation from its execution, providing flexibility in how and when payments are processed. It supports extensible and maintainable transaction handling by isolating payment logic from user interface components, and enables advanced capabilities such as queuing, logging, and future implementation of undo or audit functionality. This design not only ensures clear separation of concerns but also facilitates compliance, traceability, and robust transaction management in high-volume environments.



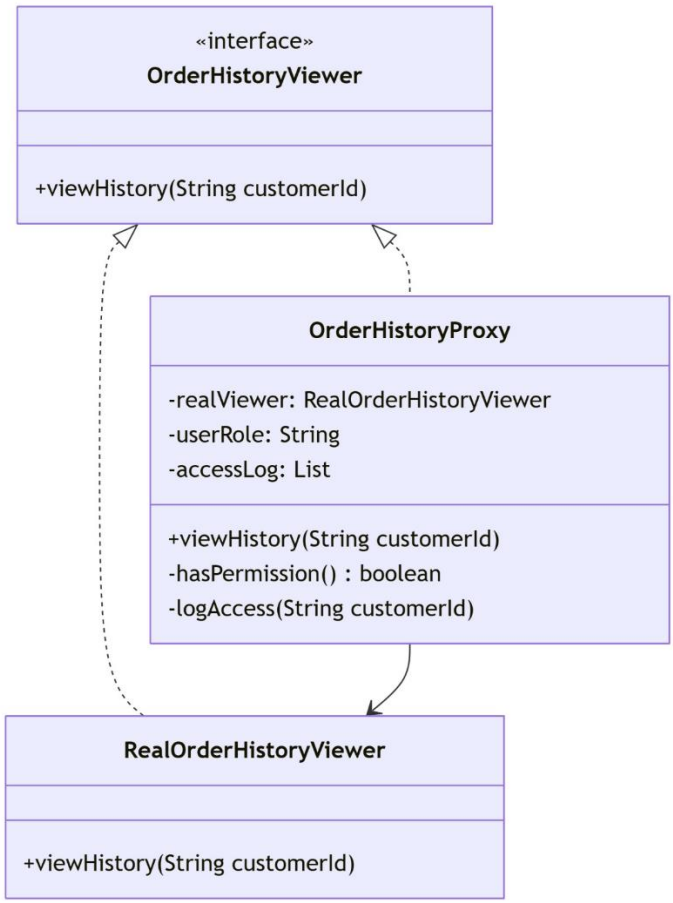
Gloria Traja

Proxy Pattern – Customer Order History Viewer

In this design, the Proxy Pattern is implemented to control access to sensitive customer order history data. The OrderHistoryViewer interface defines the viewHistory(customerId) method, and the OrderHistoryProxy acts as an intermediary between the client and the actual RealOrderHistoryViewer. The proxy checks for user role permissions before granting access and maintains a log of all view requests to support auditing and accountability. This

design encapsulates access control and usage tracking while still allowing authorized users full functionality through the real object.

The Proxy Pattern here provides: • Security, by limiting access based on user roles (e.g., only managers can view history), • Monitoring, by logging access attempts, • Separation of concerns, by offloading permission logic from the core data viewer.



Singleton Pattern – Menu Management

In this design, the Singleton Pattern is implemented through the MenuManager class to ensure that only one instance of the menu system exists throughout the entire application.

The constructor is made private, preventing external instantiation, and a static method getInstance() provides a global access point. This design guarantees that all components—such as waiters, kitchen staff, and online ordering systems—interact with the same centralized menu, maintaining consistency in menu items. Additionally, MenuManager encapsulates methods like addItem(), removeItem(), updateItem(), and getMenu() to support controlled and synchronized modifications. By using the Singleton Pattern here, the system: • Avoids duplication of menu data, • Maintains a single source of truth, • Simplifies coordination of real-time updates across the restaurant.