**MEDIATOR PATTERN**

**CHATBOT**

The Chatbot class functions as a mediator between clients and landlords, as well as property managers and administrative staff, bridging the gap between users and a variety of backend subsystems like FAQ, lease management tools, and event scheduling systems. Instead of allowing users to interact directly with these individual services, all communication and task requests are funneled through the chatbot, creating a clean separation of concerns. It maintains important attributes like chatAnswersLimit, isVoiceEnabled, and conversationHistory, which help regulate and personalize the interaction flow. It also provides multi-functional methods such as performTasks(), answerFAQ(), and explainsLeaseAgreement(), each designed to abstract backend complexity and unify access to diverse features. This architectural design ensures that any updates or changes to the underlying systems can be made independently without disrupting the user interface or user experience. By owning the dialogue context and managing which system responds to which user query, the chatbot reduces coupling, simplifies the user experience, and acts as an orchestrator of all front-end interactions. The Chatbot class fully embodies the Mediator Design Pattern by offering a one-to-many communication pipeline, coordinating several subsystems through a single, centralized interface.

**SUPPORTSTAFF**

The SupportStaff class is a mediator between various roles within the system, primarily coordinating communication and responsibility flow between tenants, supervisors, and task management subsystems. By owning key attributes like assignedTasks and assignedTickets, it becomes the central authority for managing and distributing operational workloads. This design avoids direct interactions between tenants and technicians or between supervisors and every task, thereby enforcing a clean separation of concerns. Through methods such as assignMaintenanceTask(), the class translates tenant needs or chatbot mediated service requests into actionable assignments for field workers or internal technicians. The method assistTenant() acts as a service layer, shielding tenants from internal processes and providing a simple interface for support, or by reportToSupervisor() and logCommunication() ensure that upper management is kept informed without needing to be directly embedded in routine operations. In an enterprise setting, this centralized handling streamlines workflows and supports scalable coordination among various layers of staff and users

**PROPERTYMANAGER**

The PropertyManager class is used for interactions between tenants, landlords, and properties, specifically managing communication, scheduling, and coordination tasks. This class doesn’t merely serve a passive management role it actively orchestrates processes and enforces interaction protocols among various components. Through attributes like assignedProperties, which define the scope of control, and messages, which record communication history, the class asserts ownership over the context it mediates. Its methods, such as scheduleViewing(property, date) and coordinateMaintenanceRequest(property), ensure that requests originating from tenants are appropriately routed and executed without exposing internal structures like landlord calendars or maintenance logs. Tenants don’t directly interact with landlords they instead channel their requests through the PropertyManager, which translates, validates, and dispatches these actions according to system rules. Similarly, the respondToTenantMessage() method provides a buffered, policy-compliant way for communication to occur, while generateFinancialReport() delivers insights to landlords without allowing them to directly access transactional data, maintaining abstraction. This tightly encapsulated mediation ensures that each party in the system stays loosely coupled while still participating in meaningful, structured workflows.

**LEGALADVISOR**

The LegalAdvisor class assumes responsibility for all legal activities within the platform and acts a necessity in legal domains where separation and impartiality are critical. Holding a contracts list and operating methods such as reviewDoc(), verifyPropertyOwnership(), approveContractAgreement(), and resolveDispute(), the LegalAdvisor ensures that no party tenant or landlord interacts directly regarding sensitive legal matters. Instead, they channel their inputs through this legal gatekeeper. This prevents bias, reduces conflict, and maintains procedural fairness. The presence of handleEvictionProcedure() solidifies its role as a sole executor of critical legal workflows, shielding other classes from having to manage these high-risk operations. In real systems, legal mediation is crucial to ensure documentation complies with regulations, and actions like eviction or contract approval follow lawful paths. Centralizing these responsibilities in one class enhances compliance, auditability, and consistency across the system.

**FINANCEOFFICER**

The FinanceOfficer class clearly plays the role of a financial mediator, centralizing the control and processing of monetary interactions across the system. With critical attributes like payments and financial forecasting capabilities, this class takes on responsibilities that would otherwise require complex entanglement between tenants, payment systems, and property management. The methods it offers generateFinancialReport(), trackExpense(), processRefund(), and budgetForecast()these all represent different layers of financial communication and computation. Tenants don't handle their own refunds directly instead, they submit a request, and the FinanceOfficer processes and approves it. Likewise, reporting and forecasting are carried out in this single location, ensuring that financial logic remains consistent and encapsulated. By acting as the financial brain of the system, this class reduces redundancy, improves the reliability of financial information, and isolates financial change logic from other system areas. For example, if new refund rules or tax policies are introduced, only the FinanceOfficer needs to be updated, keeping all consumers of financial data (tenants, landlords, managers) decoupled and stable.