Implementation of Chain of Responsibility Design Pattern- Ester Shumeli

Behavioral Design Pattern – Chain of Responsibility

In the Furniture Management System, the Chain of Responsibility Design Pattern is applied to streamline the handling of pending payment issues. Instead of writing multiple `if-else` conditions scattered across the system, the pattern delegates the responsibility of handling payment problems across a chain of role-based handlers.

- A centralized abstract class `PaymentHandler` defines a `handle(payment: Payment)` method and maintains a reference to the next handler in the chain via `setNext(handler: PaymentHandler)`.
- Four concrete handlers — `SalesAgentHandler`, `HRManagerHandler`, `CFOHandler`, and `AdminHandler` — extend this base class. Each handler examines the payment's `issueType` and decides whether it can resolve it or should forward it to the next handler in the chain.
- The pattern decouples the sender (`Payment`) from the receiver logic, promoting cleaner code, centralized control, and easier future expansion.

This design ensures that each department handles only the issues it is equipped to deal with, while unresolved issues are escalated to the next responsible unit. It supports the **Open/Closed Principle**, allowing new handler roles to be added without altering existing logic.

Handler Breakdown

1. SalesAgentHandler
Role: First-level handler — checks if the payment issue is related to sales commission or client-side payment delays.
Handles cases like: Unpaid sales commission for completed orders/ Delays in client payment impacting agent salary/ Adjustments due to returned products
Forwards to next if:
-Issue is unrelated to sales
-Agent lacks authority to resolve (e.g., client bankruptcy)

2. HRManagerHandler
Role: Second-level handler — focuses on employee status or contract-related payment problems.

Handles cases like: Employee marked inactive or terminated/ Contract not signed or expired/ Misclassified leave status affecting salary
Forwards to next if:
-All HR data is valid
-Payment issue seems purely financial

3. CFOHandler
Role: Third-level handler — deals with payroll, finance errors, and budget constraints.
Handles cases like: Payroll miscalculations/ Department over budget/ Incorrect deductions (e.g., tax, insurance)
Forwards to Admin if:
-Finance systems show no errors
-Issue falls outside CFO's scope (e.g., system bug)

4. AdminHandler
Role: Final fallback — manually reviews and escalates unresolved payment issues.
Handles cases like:
-Unclassified or ambiguous payment issues
-System-level failures or overrides
-Logging and notifying IT or supervisors
Typically ends the chain by:
-Alerting developers or admins
-Logging issue in audit database
-Manually resolving or deferring the payment

Example Scenario Walkthrough

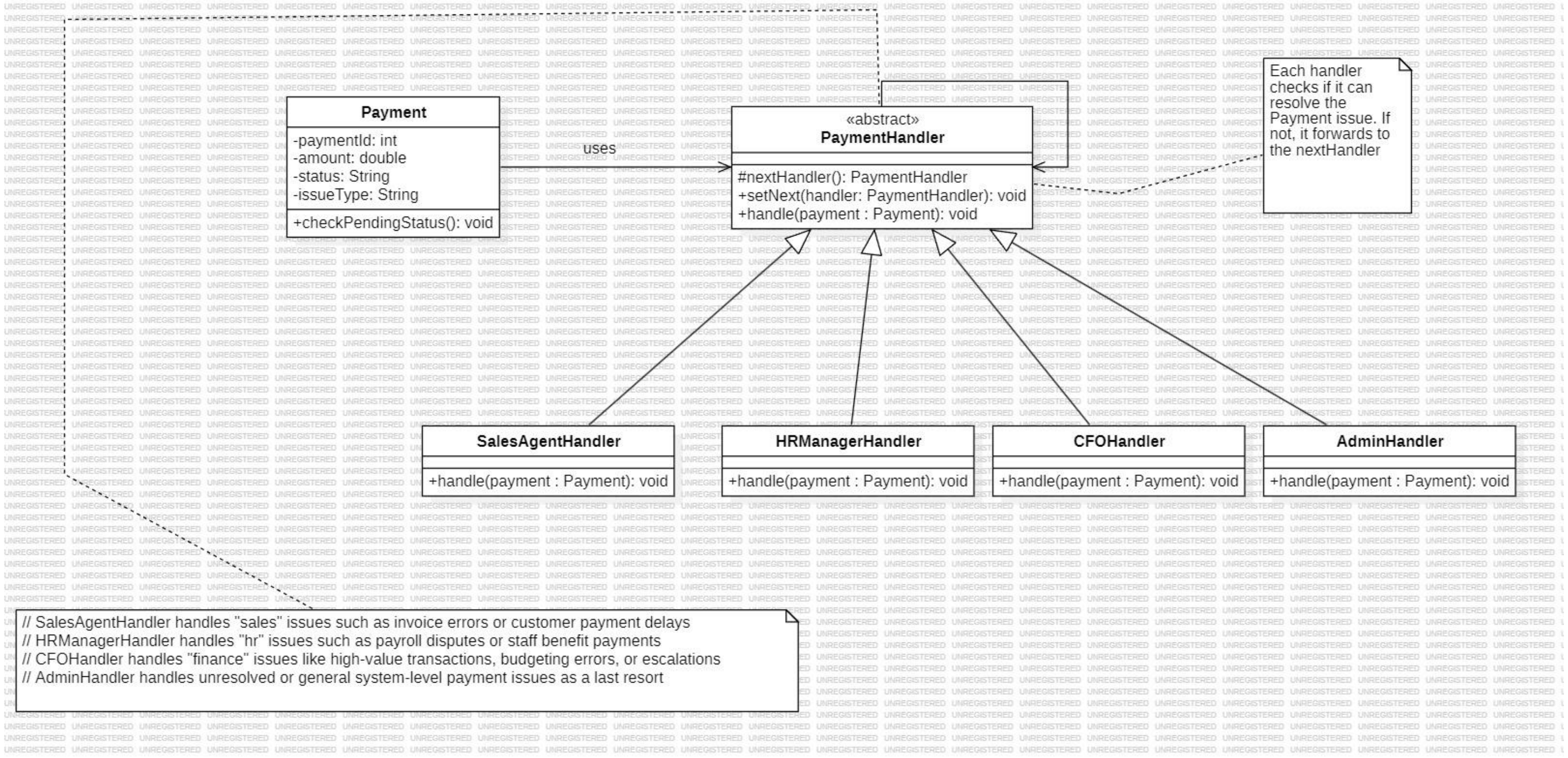
Case 1: *A sales agent reports a missed commission*
→ SalesAgentHandler: ✅ Resolves it

Case 2: *An intern hasn't received payment*
→ SalesAgentHandler: ✗
→ HRManagerHandler: ✅ Detects expired contract → Resolves

Case 3: *A finance officer's net salary is wrong*
→ SalesAgentHandler: ✗
→ HRManagerHandler: ✗
→ CFOHandler: ✅ Finds tax deduction error → Resolves

Case 4: *A payment is missing with no clear trace*
→ SalesAgentHandler: ✗
→ HRManagerHandler: ✗
→ CFOHandler: ✗
→ AdminHandler: ✅ Logs the issue, notifies admin, escalates

This pattern greatly improves maintainability and modularity by giving each department autonomy over the issues they understand best, while seamlessly escalating complex problems. It prevents tight coupling and scattered logic, ensuring every role acts only within its domain of responsibility.



Payment
-paymentId: int
-amount: double
-status: String
-issueType: String
+checkPendingStatus(): void
uses
«abstract»
PaymentHandler
#nextHandler(): PaymentHandler
+setNext(handler: PaymentHandler): void
+handle(payment : Payment): void
Each handler checks if it can resolve the Payment issue. If not, it forwards to the nextHandler
SalesAgentHandler
+handle(payment : Payment): void
HRManagerHandler
+handle(payment : Payment): void
CFOHandler
+handle(payment : Payment): void
AdminHandler
+handle(payment : Payment): void
// SalesAgentHandler handles "sales" issues such as invoice errors or customer payment delays
// HRManagerHandler handles "hr" issues such as payroll disputes or staff benefit payments
// CFOHandler handles "finance" issues like high-value transactions, budgeting errors, or escalations
// AdminHandler handles unresolved or general system-level payment issues as a last resort