Endri Baku
Sidrit Isufi

The Strategy Pattern: Justification

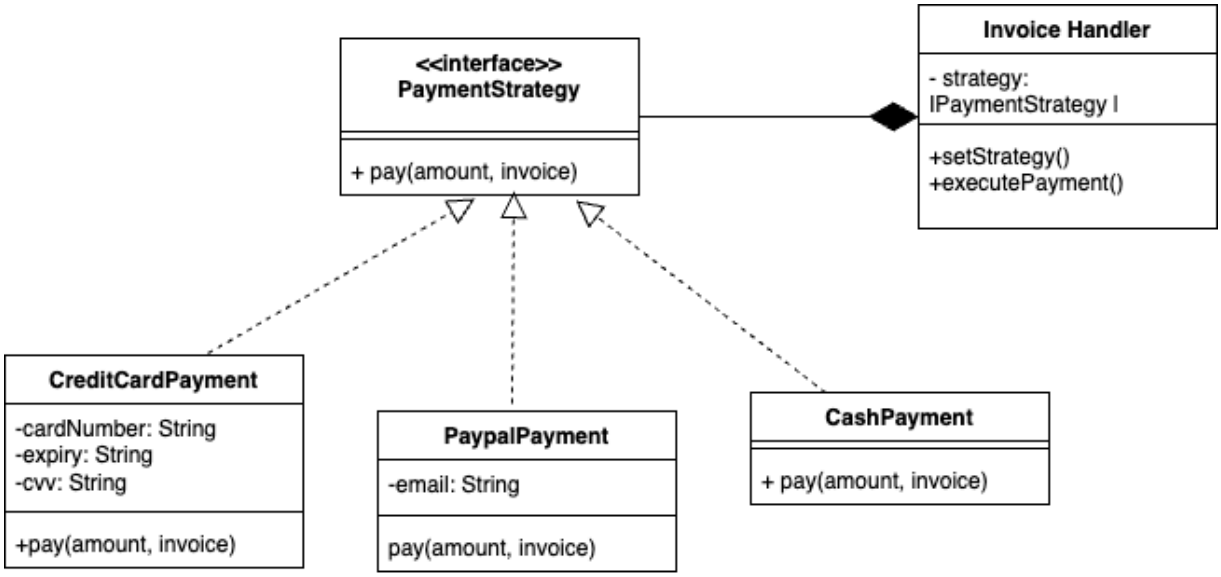
Pattern Name: Strategy Pattern

Intent of the Pattern: The Strategy Pattern is used to define a family of algorithms, encapsulate each one, and make them interchangeable. This pattern allows the behavior of an object to be selected at runtime.

Context in the Hotel Management System: In our Hotel Management System, the Strategy Pattern is implemented for the payment processing mechanism. This allows us to dynamically choose between multiple payment methods when processing transactions such as guest invoices.

Participants and Structure:

- **IPaymentStrategy (Interface):** Declares the `pay()` method that each payment strategy implements.
- **Concrete Strategies:**
  - `BankTransferPayment`
  - `CashPayment`
  - `CardPayment`
  - (Optionally) `CryptoPayment (would be easy to add)`
- **Context Class:**
  - `PayrollHandler` or `InvoiceHandler` which maintains a reference to an IPaymentStrategy and uses it to execute payment.



Justification:

- **Flexibility:** Easily switch between different payment methods without changing the handler logic.
- **Maintainability:** Payment logic is isolated within separate strategy classes.
- **Extensibility:** New payment methods can be added without modifying existing code.
- **Cleaner Code:** Avoids complex conditionals or switch statements for handling different payment types.

Conclusion: The Strategy Pattern is an excellent fit for our Hotel Management System's payment module, as it cleanly supports multiple dynamic behaviors (payment methods) and ensures that our codebase remains modular and scalable.