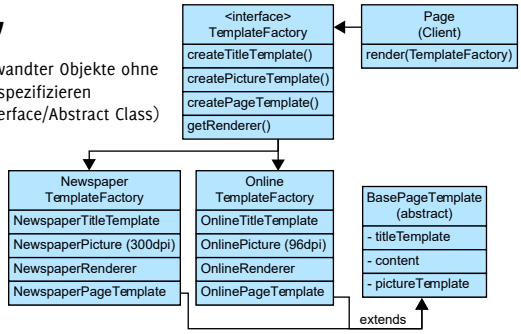
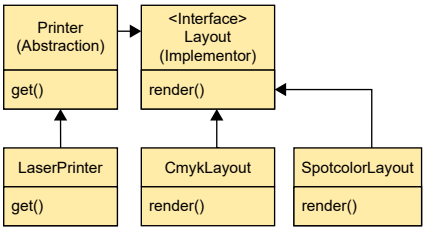
Abstract Factory

- Erzeugt Familien verwandter Objekte ohne konkrete Klassen zu spezifizieren
- Abstract Factory: (Interface/Abstract Class) zur Erzeugung verwandter Objekte
- Client: nutzt Factory abstrakt



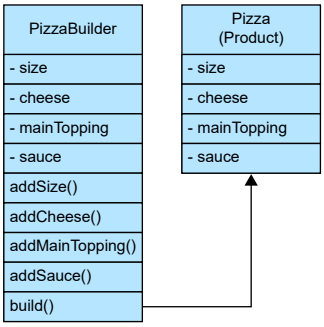
Bridge

- Trennt die Abstraktion von ihrer Implementierung
- Abstraction: abstrakte Basisklasse
- Refined Abstractions: konkrete Abstraktionen, die die Basisklasse erweitern
- Implementor Interface: Schnittstelle für Strategien
- Concrete Implementors: konkrete Darstellungsvarianten



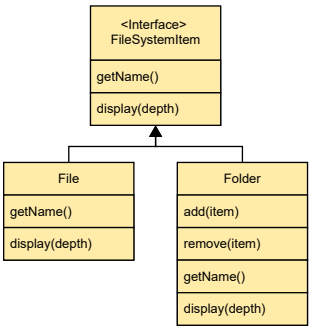
Builder

- Trennt Konstruktion eines komplexen Objekts vom eigentlichen Objekt
- Builder: enthält Zustand (+ fluent API)
- Product: unveränderbares Ergebnis
- Client: Code, der mit dem Builder das Objekt erzeugt
- Director: definiert vorgefertigte Baupläne



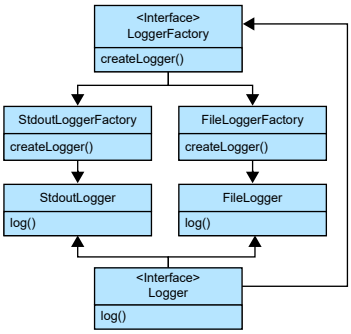
Composite

- Objekte zu Baumstrukturen zusammenzufassen, um einzelne Objekte und Objektgruppen einheitlich zu behandeln
- Component (Interface): definiert eine gemeinsame Schnittstelle
- Leaf: repräsentiert „Blatt“ ohne Kinder
- Composit: repräsentiert ein Element, dass Kinder enthalten kann



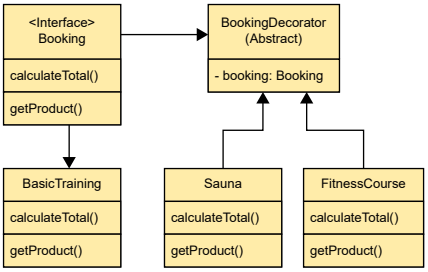
Factory Method

- Kapselt Objekt-Erstellung
- Erlaubt Unterklassen, eigene Produkte zu liefern
- Abstraktes Produkt: Interface
- Creator (Factory): Interface
- Client: nutzt Factories, ohne Wissen über konkreteKlasse



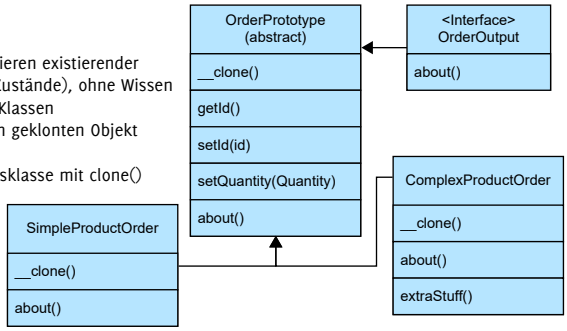
Decorator

- Dynamisches Hinzufügen von Verhalten zu einem Objekt
- Component (Interface): Definiert die Schnittstelle für zu dekodierende Objekte
- ConcreteComponent: konkrete Implementierung
- Decorator (Abstrakt): Referenz auf Objekt
- ConcreteDecorators: Dekoratoren, die das Verhalten erweitern



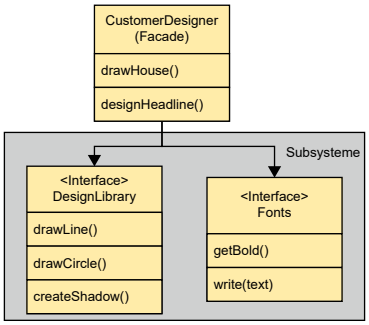
Prototype

- Ermöglicht Kopieren existierender Objekte (inkl. Zustände), ohne Wissen über konkrete Klassen
- Änderungen am geklonten Objekt möglich
- Prototype: Basisklasse mit clone() (abstract)



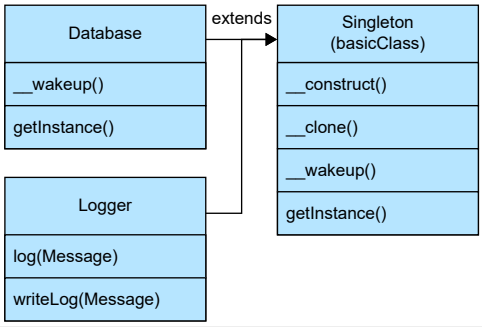
Facade

- Einheitliche Schnittstelle zu einer Gruppe von Schnittstellen in einem Subsystem
- Subsysteme Interfaces oder komplexe Klassen
- Reduziert Komplexität für den Client



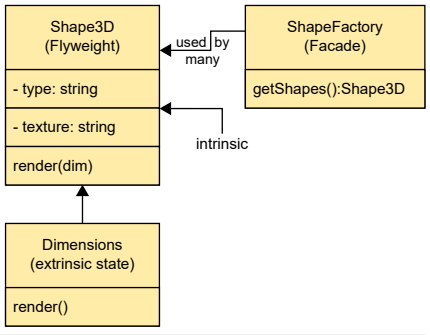
Singleton

- Zentrale Verwaltung der Singleton-Instanz
- Enthält Logik, um sicherzustellen, dass nur eine Instanz pro Subklasse existiert
- Verhindert durch Konstruktor, clone() und wakeup() das Erzeugen zusätzlicher Instanzen



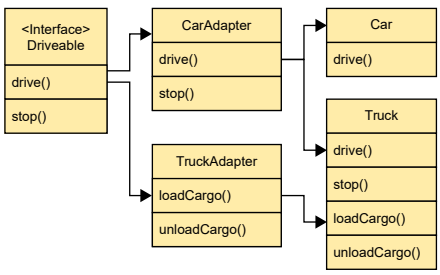
Flyweight

- Minimiert Speicherverbrauch: ähnliche Objekte teilen gemeinsame Zustände
- Intrinsisch: sich häufig wiederholende Teile
- Diese einmal erzeugt und wiederverwendet
- Extrinsisch: variierende Daten, von außen übergeben
- Flyweight: nur gemeinsam nutzbare, unveränderliche Daten
- Factory verwaltet und cached Flyweights



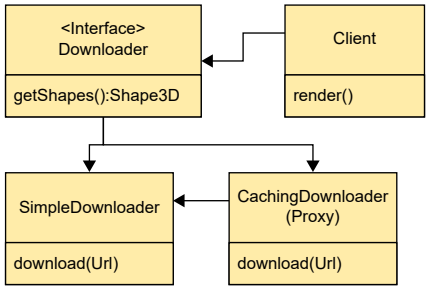
Adapter

- Zusammenarbeit von Klassen mit inkompatiblen Schnittstellen
- Client Interface: Zielinterface, vom Client erwartet wird
- Adaptee: ursprünglichen Klassen mit ggf. inkompatibler Schnittstelle
- Adapter: passen die Adaptee-Methoden an das Interface an



Proxy

- Stellvertreter-Objekt, das Zugriff auf anderes Objekt kontrolliert
- Subject: Interface
- Real Subject: konkreter Service
- Client nutzt Subject



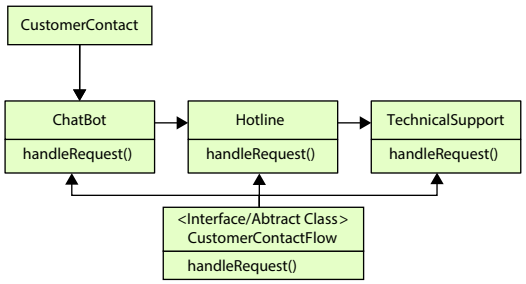
Creational Pattern

Structural Pattern

by Sven Duge, 2025

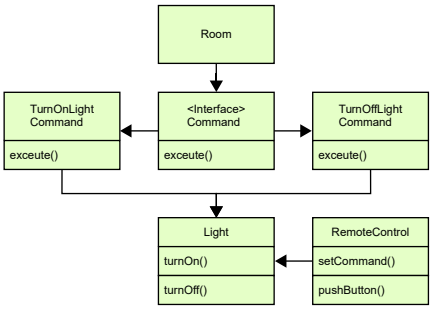
Chain of Responsibility

- Verkettung von Handlern
- Abstrakte Basisklasse mit Verkettungslogik
- Jeder Handler kapselt die Verarbeitung einer Anfrage



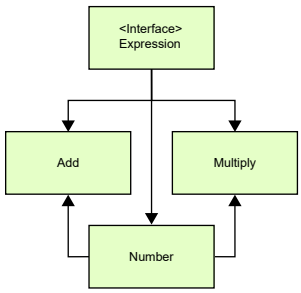
Command

- Command Interface: Enthält execute()
- Concrete Command: Kapseln Aktion & Empfänger
- Receiver: Kennt die tatsächliche Logik
- Invoker: Führt execute() des Commands aus
- Client: Setzt alles zusammen und startet Befehle



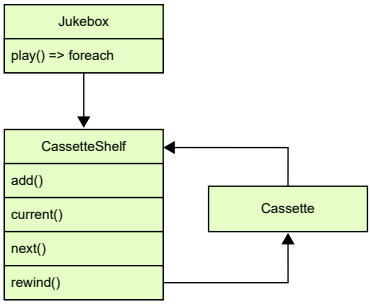
Interpreter

- Abstrakte Regel Interface: definiert interpret()
- Terminal-Expression: enthält den konkreten Wert
- Non-Terminal-Expressions: enthalten rekursive Auswertung
- Rekursive Struktur: bringt alles zusammen



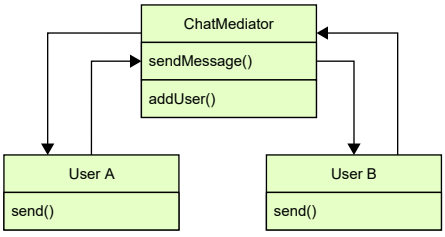
Iterator

- Element: Einfache Datenklasse
- Collection: Kapselt Array + Iterationslogik
- Iterator: Implementiert Iterator direkt
- Zugriff: wie erwartet mit foreach



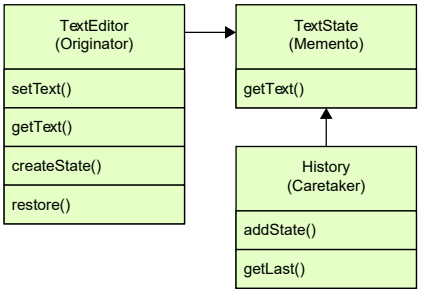
Mediator

- Mediator-Interface: definiert API für Kommunikation
- Concrete Mediator: verwaltet und verteilt Nachrichten
- Colleague / Teilnehmer: kennt den Mediator und nutzt ihn
- Entkoppelte Kommunikation: User reden nicht direkt miteinander



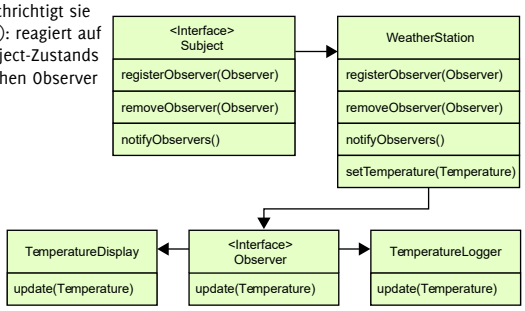
Memento

- Bewahrt den Zustand, ohne die Details nach außen zu geben
- Kein direkter Zugriff auf innere Struktur
- Originator: erstellt Memento oder stellt ihn wieder her
- Memento: enthält gespeicherten Zustand
- Caretaker: verwaltet die Zustände



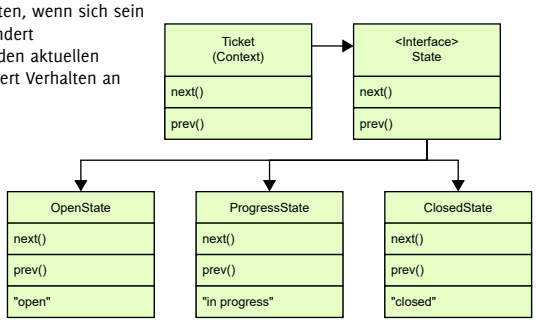
Observer

- Subject (Publisher): hält Liste von Observers und benachrichtigt sie
- Observer (Subscriber): reagiert auf Änderungen des Subject-Zustands
- Lose Kopplung zwischen Observer und Subject



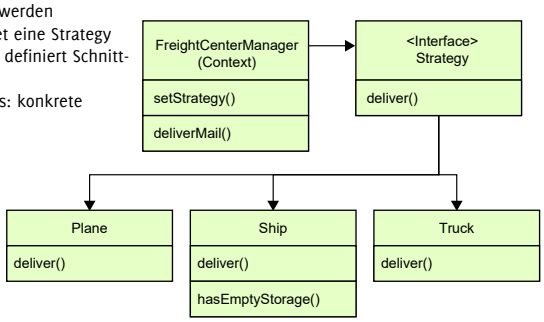
State

- Objekt kann Verhalten, wenn sich sein interner Zustand ändert
- Context: verwaltet den aktuellen Zustand und delegiert Verhalten an diesen
- State Interface: definiert gemeinsame Methoden für alle Zustände
- Concrete States: konkrete Implementierungen



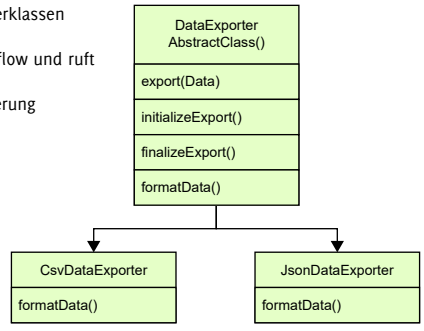
Strategy

- Verhalten eines Objekts kann zur Laufzeit geändert werden
- Context: verwendet eine Strategy
- Strategy Interface: definiert Schnittstelle
- Concrete Strategies: konkrete Implementierung



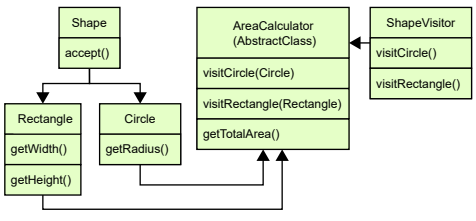
Template Method

- Template gibt den Ablauf vor, Unterklassen dürfen einzelne Schritte anpassen
- Abstract Class: definiert den Workflow und ruft die variablen Schritte auf
- Subklassen: konkrete Implementierung



Visitor

- Ermöglicht neue Operationen, ohne Klassen zu ändern
- Concrete Visitor: implementiert Logik
- Concrete Elements: konkrete Implementierung
- Visitor Interface: definiert Besuchsmethoden für jedes konkrete Element
- Element-Interface: definiert Elemente



Behavioral Pattern

by Sven Duge, 2025