

GoF-Entwurfsmuster – Spickzettel für den Vortrag

GoF Design Patterns · ca. 45 Minuten · Demo: Nordfracht Logistik GmbH (3 Konsolenprojekte, 9 Muster)

Ablauf

Min	Block	Stichpunkte zum Ablesen	Überleitung
0–4	Einstieg	Frage: „Wer hat schon ein Entwurfsmuster benutzt?“ → Alle: foreach = Iterator, event = Observer, ICommand = Command	„Sie nutzen Muster längst – heute geben wir ihnen Namen.“
4–10	Was & woher	Definition · 4 Elemente (Name, Problem, Lösung, Konsequenzen) · Alexander 1977 → GoF 1994	„Die GoF haben 23 Muster in drei Schubladen sortiert.“
10–14	Einordnung & Prinzipien	Ebenen: Architektur (MVVM!) / Entwurf / Idiom · 3 Kategorien · Klassen- vs. Objektmuster · 2 GoF-Prinzipien	„Beginnen wir dort, wo Objekte entstehen.“
14–22	Erzeugungsmuster	Singleton · Factory Method · Builder · Demo 1	„Objekte sind da – wie stecken wir sie zusammen?“
22–30	Strukturmuster	Adapter · Decorator · Facade · Demo 2 (Klassenexplosion-Frage)	„Und wie arbeiten sie zusammen?“
30–38	Verhaltensmuster	Strategy · Observer · Command · Demo 3 (Beobachter live ergänzen)	„Klingt gut – kann man es übertreiben? Ja.“
38–42	Kritik & Prüfung	Overengineering · Singleton als Antimuster · moderne Sprachen machen Muster überflüssig · AP2: Muster in UML/Code erkennen	„Und was kommt nach der GoF?“
42–45	Ausblick & Fazit	Architektur-, Enterprise-, Cloud-Muster · Fragen	„Muster sind Vokabeln. Wer sie kennt, redet schneller über Entwürfe.“

Zeitstrahl

Jahr	Ereignis
1977	Architekt Christopher Alexander : „A Pattern Language“ – 253 Muster für Städte und Gebäude
1987	Erste Übertragung der Musteridee auf Software
1994	„ Gang of Four “: Buch „Design Patterns“ – 23 Muster in 3 Kategorien
1996	Katalog der Architekturmuster (Schichten, MVC ...) · erste Kritik: in dynamischen Sprachen sind viele Muster überflüssig
ab 2000	Muster für Unternehmensanwendungen (Repository, Unit of Work) und später Cloud-Muster

Die 23 GoF-Muster nach Kategorie

Erzeugungsmuster (5)	Strukturmuster (7)	Verhaltensmuster (11)
Wie entstehen Objekte? Abstrakte Fabrik – Abstract Factory Erbauer – Builder ★ Fabrikmethode – Factory Method ★ Prototyp – Prototype Singleton ★	Wie werden Objekte zusammengesetzt? Adapter ★ Brücke – Bridge Kompositum – Composite Dekorierer – Decorator ★ Fassade – Facade ★ Fliegengewicht – Flyweight Stellvertreter – Proxy	Wie arbeiten Objekte zusammen? Zuständigkeitskette – Chain of Responsibility Befehl – Command ★ · Interpreter · Iterator Vermittler – Mediator · Memento Beobachter – Observer ★ · Zustand – State Strategie – Strategy ★ Schablonenmethode – Template Method Besucher – Visitor

★ = im Vortrag mit Demo · Klassenmuster (Vererbung, Compilezeit): Factory Method, Klassen-Adapter, Interpreter, Template Method – alle anderen sind Objektmuster.

Die 9 Kernmuster

Muster (EN / DE)	Problem	Lösung	Erkennen am
Singleton Singleton	genau eine Instanz, global erreichbar	privater Konstruktor + statische Instanz	<code>private Konfiguration(), static Instanz</code>
Factory Method Fabrikmethode	Ablauf fix, erzeugter Typ variiert	abstrakte Erzeugungsmethode, Unterklassen entscheiden	<code>abstract ... Erzeuge...() +</code> Unterklassen
Builder Erbauer	viele optionale Parameter	schrittweiser Aufbau, <code>Build()</code> prüft	Methodenkette <code>.Mit...().Build()</code>
Adapter Adapter	Schnittstelle passt nicht	Übersetzerklasse implementiert Ziel-Interface	implementiert A, hält B
Decorator Dekorierer	frei kombinierbare Zusätze	Hülle um gleiches Interface	implementiert X und bekommt X im Konstruktor
Facade Fassade	komplexes Subsystem	eine einfache Einstiegsklasse	wenige Methoden, viele Objekte dahinter
Strategy Strategie	austauschbarer Algorithmus, <code>switch</code> -Wüste	Algorithmus in eigene Klassen	Interface mit 1 Methode, Kontext hält Referenz
Observer Beobachter	viele sollen bei Änderung reagieren	Subjekt verwaltet Liste von Beobachtern	<code>Anmelden()/Abmelden()</code> + Schleife
Command Befehl	Aktion speichern, rückgängig machen	Aktion als Objekt mit <code>Ausfuehren/Rueckgaengig</code>	Interface <code>Execute/Undo</code> + Stack

Prinzipien hinter den Mustern

- **Programmiere gegen eine Schnittstelle, nicht gegen eine Implementierung.** (GoF)
- **Ziehe Objektkomposition der Klassenvererbung vor.** (GoF) – Decorator statt Klassenexplosion.
- **Kapsle, was variiert.** – Strategy kapselt den Tarif, Factory Method den erzeugten Typ.
- **Offen für Erweiterung, geschlossen für Änderung** (Open/Closed, SOLID) – neuer Beobachter, `Sendung` bleibt unverändert.

Demo-Checkliste

✓	Schritt	Was zeigen / sagen
☐	Demo 1	<code>new Konfiguration()</code> einkommentieren → Compilerfehler · Builder: Konstruktor-Albtraum aus dem Kommentar vorlesen
☐	Demo 2	Decorator-Zeile zeigen · Frage: „Wie viele Klassen mit Vererbung?“ → 8 bei 3 Zusätzen, 16 bei 4
☐	Demo 3	Live <code>LogBeobachter</code> ergänzen, ohne <code>Sendung</code> zu ändern · Command: zweimal Rückgängig · Brücke MVVM: Observer = <code>INotifyPropertyChanged</code> , Command = <code>ICommand</code>

Kritik & mögliche Fragen

Frage	Antwort
Muss ich alle 23 kennen?	Nein. 8–10 sicher, den Rest wiedererkennen. Wichtiger: das Problem hinter dem Muster.
Ist Singleton schlecht?	Oft ja: globaler Zustand, schwer testbar. Besser: eine Instanz per Dependency Injection.
Adapter vs. Decorator vs. Proxy?	Adapter ändert die Schnittstelle, Decorator erweitert Verhalten, Proxy kontrolliert Zugriff.
Strategy vs. State?	Gleiche Struktur. Strategy: Aufrufer wählt. State: Objekt wechselt selbst je nach Zustand.
Factory Method vs. Abstract Factory?	Factory Method: eine Methode, Vererbung. Abstract Factory: Objekt erzeugt ganze Produktfamilie.
Wann KEIN Muster?	Wenn es das Problem (noch) nicht gibt. Muster nachträglich einführen (Refactoring) ist normal.