HSLU Hochschule Luzern

Verteilte Systeme und Komponenten

Entwurfsmuster

Roland Gisler

FH Zentralschweiz

Inhalt

- Einführung - Was sind Entwurfsmuster?
- Gliederung der Entwurfsmuster in Kategorien
- Ausgewählte, einfache Beispiele, Teil 1
- Einsatz von Entwurfsmustern
- Ausgewählte, einfache Beispiele, Teil 2
- Ergänzende Hinweise zu Entwurfsmustern
- Zusammenfassung und Quellen

Lernziele

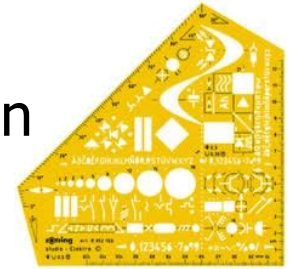
- Sie verstehen die Vorteile beim Einsatz von Entwurfsmustern.
- Sie kennen verschiedene, ausgewählte Entwurfsmuster.
- Sie können konkrete Entwurfsmuster auswählen und gezielt einsetzen.

Einführung

Entwurfsmuster

- "Elemente wiederverwendbarer, objektorientierter Software."

oder



- "Bewährte objektorientierte Entwürfe (Schablonen) für ein wiederkehrendes Entwurfsproblem."
- Massgeblich entwickelt und popularisiert von Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides†, auch bekannt als die „**G**ang **o**f **F**our" (GoF).
- Resultate
  - Rund 20 dokumentierte Entwurfsmuster.
  - Buch „Entwurfsmuster" (1995), ein echter Klassiker in der Informatik-Literatur.

    Aktuell: mitp Professional, 2015, ISBN 978-3-8266-9700-5

Erich Gamma

- Promovierte an der Universität Zürich
- Mitautor von Entwurfsmuster, Mitglied der GoF
- Mitentwickler von JUnit (mit Kent Beck)
- Aktuell:
  - Bei Microsoft als Distinguished Engineer tätig: Weiterentwicklung von Microsoft Visual Studio Code.
  - Lange als Distinguished Engineer bei Rational Software (Abteilung der IBM Software Group, mit Sitz in Zürich) tätig.
  - Leitete lange die Entwicklung der Eclipse Platform, auf der auch die populäre Eclipse JDT (IDE) basiert.

Wiederverwendung

Wiederverwendung in der SW-Entwicklung

- Wiederverwendung von bewährten Entwurfsmustern als Ziel.
- Verschiedene Arten von Wiederverwendung in der Softwareentwicklung:
  - Objekte zur Laufzeit wiederverwenden.
  - Wiederverwendung von Quellcode / Klassen.
  - Wiederverwendung von einzelnen Komponenten.
  - Einsatz von Klassen-Bibliotheken / Frameworks.
  - Wiederverwendung von Konzepten, z.B:
    - Entwurfs-, Architektur- oder Kommunikationsmuster…

Wiederverwendung von Objekten

- Wiederverwendung von Objekten in einer Software während der Laufzeit.
- Beispiele:
  - Threads ein einem **`Executor`**-Pool.
  - DB-Connection in einem Connection-Pool.
- 👍Effekt:
  - Bessere Performance, höhere Effizienz.
  - Geringerer Ressourcenbedarf.
- Herausforderung:
  - Effiziente Verwaltung der Objekte.
  - Hier können Entwurfsmuster bereits konkret helfen!

Wiederverwendung von Quellcode/Klassen

- Wiederverwendung durch
  - Copy&Paste ➔ schlecht! (vgl. Clean Code – DRY Prinzip)
  - Vererbung ➔ häufig schlecht (!).
  - Aggregation und Komposition ➔ Gut! (vgl. Clean Code – FCoI)
- 👍Effekte:
  - Geringerer Entwicklungsaufwand.
  - Geringere Fehlerrate (Klassen sind bereits umfangreich getestet).
- Herausforderungen:
  - Schnittstellen der Klassen eher fremdbestimmt.
  - Auswahl der geeigneten Klassen/Bibliotheken.
  - Lernaufwand, Wartung und Weiterentwicklung.

Wiederverwendung von Komponenten

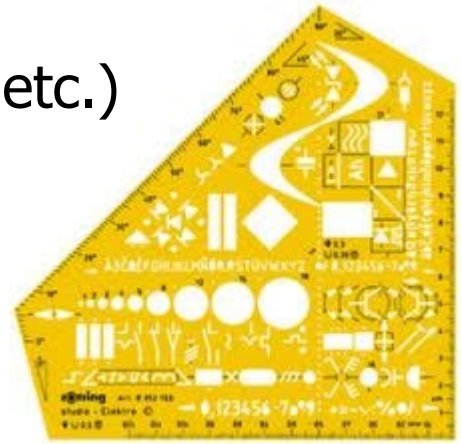
- Beispiele:
  Logging-Komponente, Jakarta EE-Beans, Corba-Komponenten etc.
- 👍Effekte:
  - Geringerer Entwicklungsaufwand, weniger Fehler.
  - Ganzheitlicherer Ansatz, Blackbox, Abstraktion.
- Herausforderungen:
  - Anforderungen an die Umgebung / Kontext.
  - Eventuell inkompatible Schnittstellen ➔ Entwurfsmuster!
  - Verwaltung der Komponenten (Konfigurationsmanagement).
  - Abhängigkeit vom Lieferanten.
  - Wartung und Weiterentwicklung.

Herausforderung der (Quellcode-)Wiederverwendung

- Wiederverwendung ist sehr gut, bringt aber auch ein paar Herausforderungen mit sich:
  - Unterschiedliche Kontexte / Fachverständnisse.
  - Unterschiedliche Technologien / Lösungsansätze.
  - Einfache Weiterentwicklung und Wartung.
  - Aufwändiges Konfigurationsmanagement.
  - Verschiedene, inkonsistente Designkonzepte.
  - Zusätzliche Abhängigkeit von Dritten.
- **Wiederverwendung von Quellcode ist und bleibt eine grosse Herausforderung!**

Alternative: Wiederverwendung von Konzepten

- Konzepte bleiben relativ konstant und stabil.
- Zusätzlich relativ breit abgestützt und erprobt, weil weitgehend Sprach- und Implementationsunabhängig!
- **Die Wiederverwendung von bewährten Entwurfsmustern ist eine sehr elegante, wirkungsvolle, unproblematische und kostensparende Form von Wiederverwendung!**
- Vergleiche:
  - Kommunikationsmuster (z.B. Handshaking)
  - Architekturmuster (z.B. C/S, Schichtung, MVC etc.)



Entwurfsmuster - Klassifikation

Klassifikation von Entwurfsmustern

- Entwurfsmuster werden primär nach Ihrem Zweck klassifiziert. Daraus sind drei Gruppen entstanden:
  - **Erzeugungsmuster** (Creational Patterns)
  - **Strukturmuster** (Structural Patterns)
  - **Verhaltensmuster** (Behavioral Patterns)
- Sekundäre Unterteilung:
  - **Klassenmuster**
    - Legen Beziehungen bereits zum Kompilierzeitpunkt fest.
  - **Objektmuster**
    - Beziehungen sind zur Laufzeit dynamisch veränderbar.

Kategorie 1: Erzeugungsmuster

- Abstrahieren die Erzeugung von Objekten.
  - Entscheidung welcher (dynamische) Typ verwendet wird.
  - Entscheid über den Zeitpunkt der Erzeugung (z.B. lazy).
  - Entscheid auf welche Art das Objekt konfiguriert wird (Kontext, Initial-Konfiguration etc.).
- Delegation der Erzeugung an ein anderes Objekt.
  - 'Fabrik'-Konzept: Man fordert einfach ein Objekt an, die Details der Instanziierung und der Konfiguration interessieren (die Nutzer*in) aber nicht.

Erzeugungsmuster - Übersicht

- Abstrakte Fabrik (Abstract Factory, Kit)
- Erbauer (Builder)
- Fabrikmethode (Factory Method, Virtual Constructor)*
- Prototyp (Prototype)*
- Einzelstück (Singleton)*

Kategorie 2: Strukturmuster

- Fassen Objekte (oder Klassen) zu grösseren oder veränderten Strukturen zusammen.

oder

- Erlauben unterschiedliche Strukturen einander anzupassen und miteinander zu verbinden.

Strukturmuster - Übersicht

- Adapter (Adapter, Wrapper)
- Brücke (Bridge, Handle/Body)
- Dekorierer (Decorator, Wrapper)
- Fassade (Facade)
- Fliegengewicht (Flyweight)
- Kompositum (Composite)*
- Stellvertreter (Proxy, Surrogate)*

Kategorie 3: Verhaltensmuster

- Beschreiben die Interaktionen zwischen Objekten.
- Legen die Kontrollflüsse zwischen den Objekten fest.
- Zuständigkeit und/oder Kontrolle delegieren.

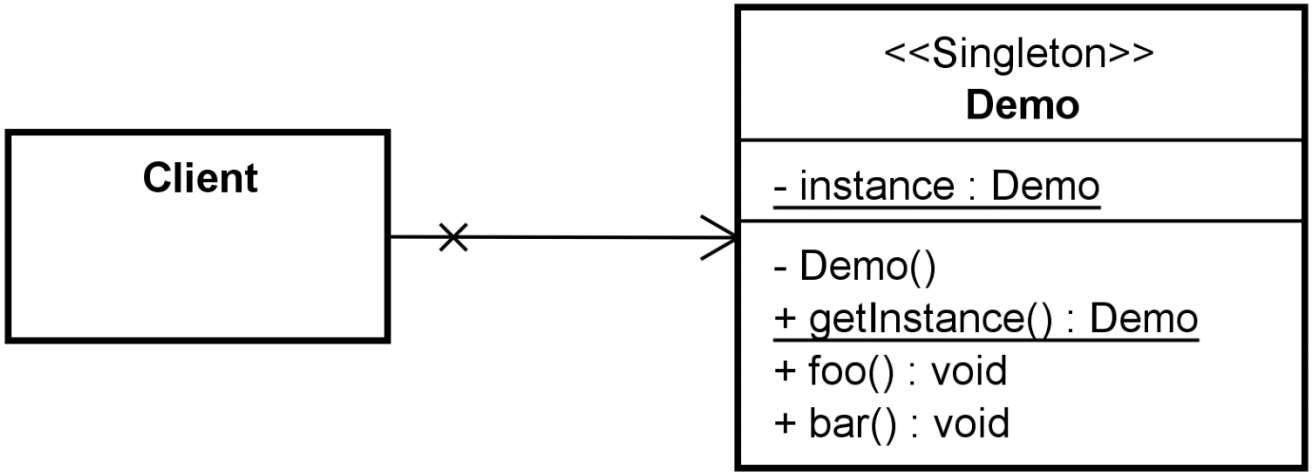
Verhaltensmuster - Übersicht

- Befehl (Kommando, Command, Action, Transaction)
- Beobachter (Observer, Dependents, Publish/Subscribe, Listener)*
- Besucher (Visitor)
- Interpreter (Interpreter)
- Iterator (Iterator, Cursor)*
- Memento (Memento, Token)
- Schablonenmethode (Template Method)
- Strategie (Strategy, Policy)
- Vermittler (Mediator)
- Zustand (State, Objects for States)*
- Zuständigkeitskette (Chain of Responsibility)

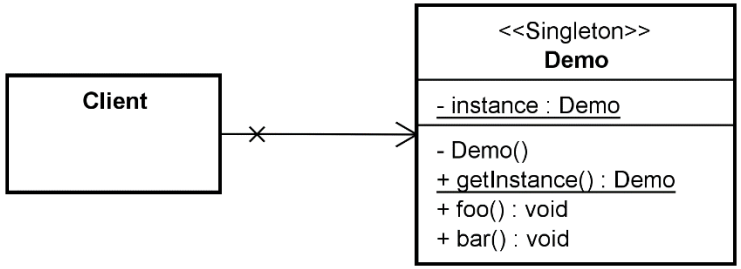
Singleton

Beispiel 1: Singleton

- Gewährleistet, dass von einer Klasse genau nur **eine einzige** Instanz (Objekt) erzeugt wird, und stellt für diese einen Zugriffspunkt zur Verfügung.



Beispiel 1: Singleton

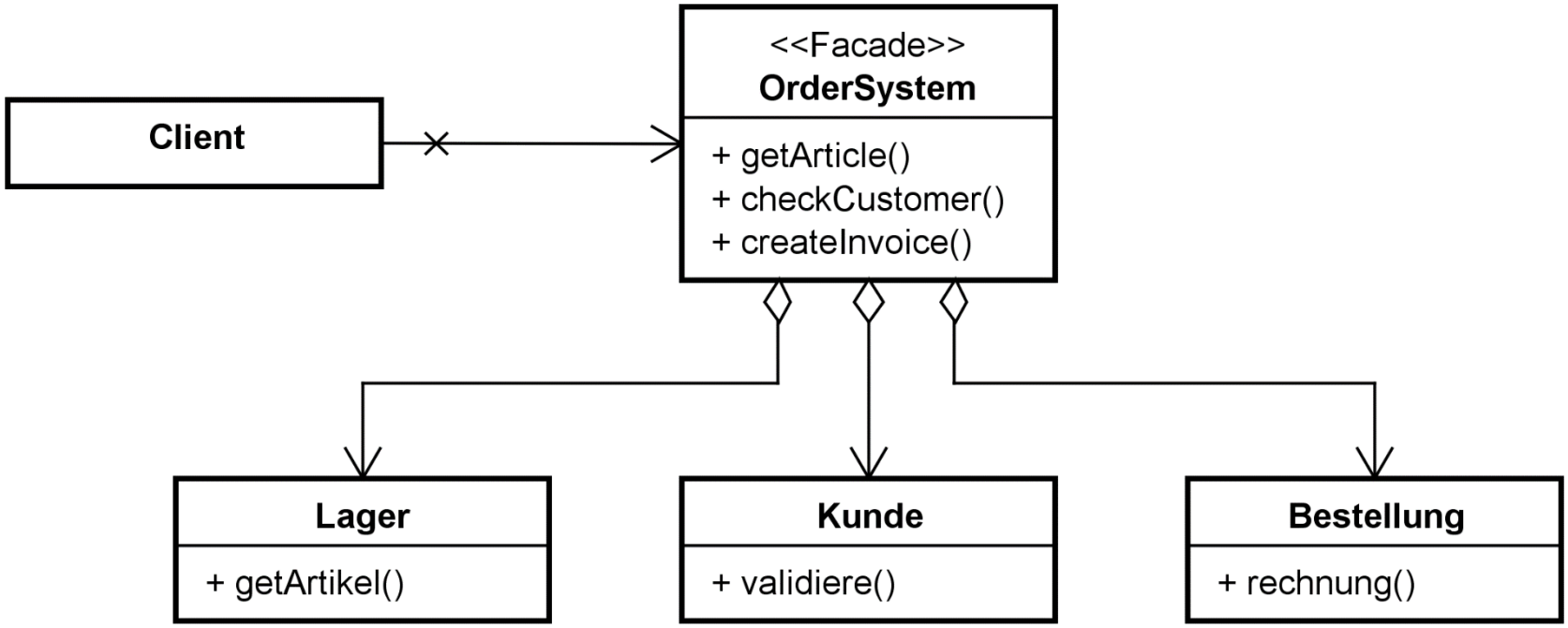


- Erzeugungsmuster, objektbasiert
- Wichtigste Eigenschaften der Implementierung:
  - Privates, statisches Attribut für Objektinstanz.
  - Öffentliche, statische Methode für Zugriff auf Objekt.
  - Privater Konstruktor (verhindert externe Instanziierung).
- Singleton hat mittlerweile einen schlechten Ruf:
  - Gamma bedauert, dieses Pattern propagiert zu haben.
  - Hauptkritik: Das Singleton führt zu einer starken Kopplung, ein späterer Austausch ist nur mit grossem Aufwand möglich.
- Empfehlung: Sehr zurückhaltend und gezielt einsetzen. Singleton **niemals** als universellen, globalen Zugriffspunkt verwenden.

Fassade

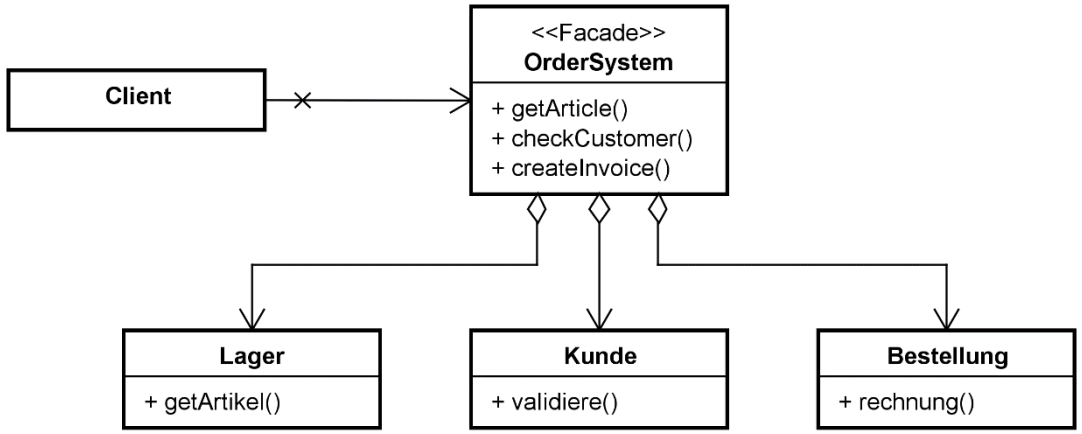
Beispiel 2: Fassade

- Stellt eine einheitliche, zusammengefasste Schnittstelle zu einer Menge von Schnittstellen mehrerer Subsysteme zur Verfügung.

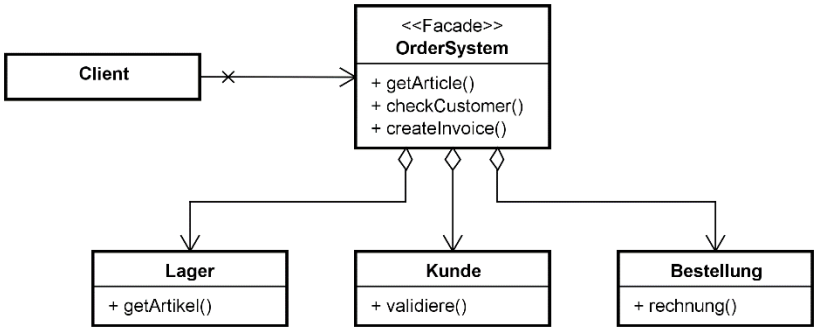


Beispiel 2: Fassade - Teilnehmer

- Fassade – Beispiel: **`OrderSystem`**
  - Weiss welche Subklassen für eine Anfrage zuständig sind und delegiert die Anfragen entsprechend weiter.
  - Sorgt für eine konsistente Namensgebung der Methoden.
  - Enthält ansonsten keine weitere Funktionalität!
- Subsystemklassen – Beispiel: **`Lager`**, **`Kunde`**, **`Bestellung`**
  - Implementieren die eigentliche Funktion.
  - Wissen nichts von der Fassade (keine Referenz).



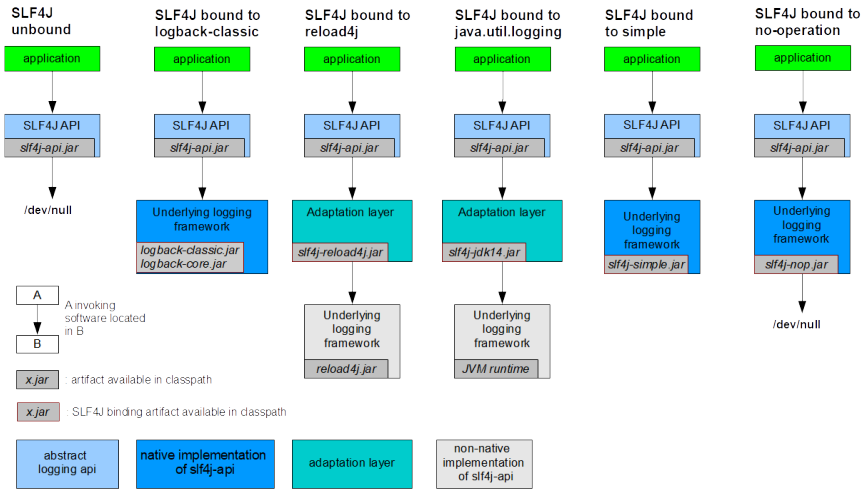
Beispiel 2: Fassade



- Strukturmuster, objektbasiert
- Motivation für Einsatz
  - Vereinfacht die Anwendung mehrerer Subsysteme.
  - Minimiert die Abhängigkeiten zu den Subsystemen.
    ➔Kopplung minimieren!
  - Einfache Austauschbarkeit eines Subsystems ermöglichen.
- Gefahr: Verkommt zum reinen Durchlauferhitzer.
- Empfehlung: Mit einer Fassade lässt sich sehr gut und einfach entkoppeln. Darauf achten, dass die Fassade nicht plötzlich wesentliche Funktionalität enthält, sie **delegiert** ausschliesslich!

Beispiel 2: Simple Logging Facade for Java (SLF4J)

- Beispiel für Einsatz des Fassaden-Patterns. https://slf4j.org
- Die Fassade bleibt einheitlich, «darunter» können sich verschiedene Logging-Frameworks verstecken.
  - LogBack implementiert SLF4J beispielsweise direkt,
  - alle anderen Frameworks benötigen ➔**Adapter**-Libraries, wobei das korrekterweise eigentlich Bridges wären.

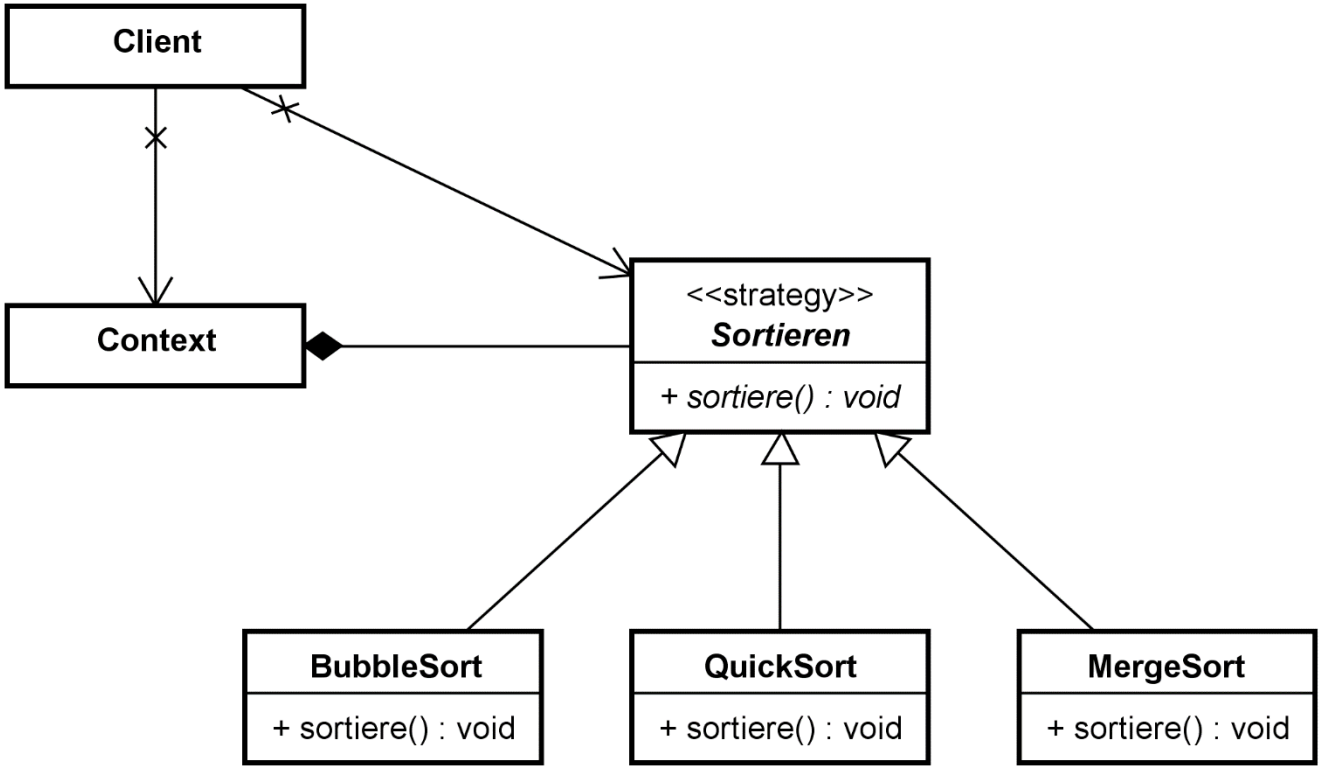


- Das konkret genutztes Logging Framework wird ausschliesslich durch entsprechendes Deployment bzw. über Classpath gesteuert!
- Ideal für Library- oder Framework-Entwicklung: Logging-Framework kann flexibel von Nutzer*in ausgewählt werden.

Strategie

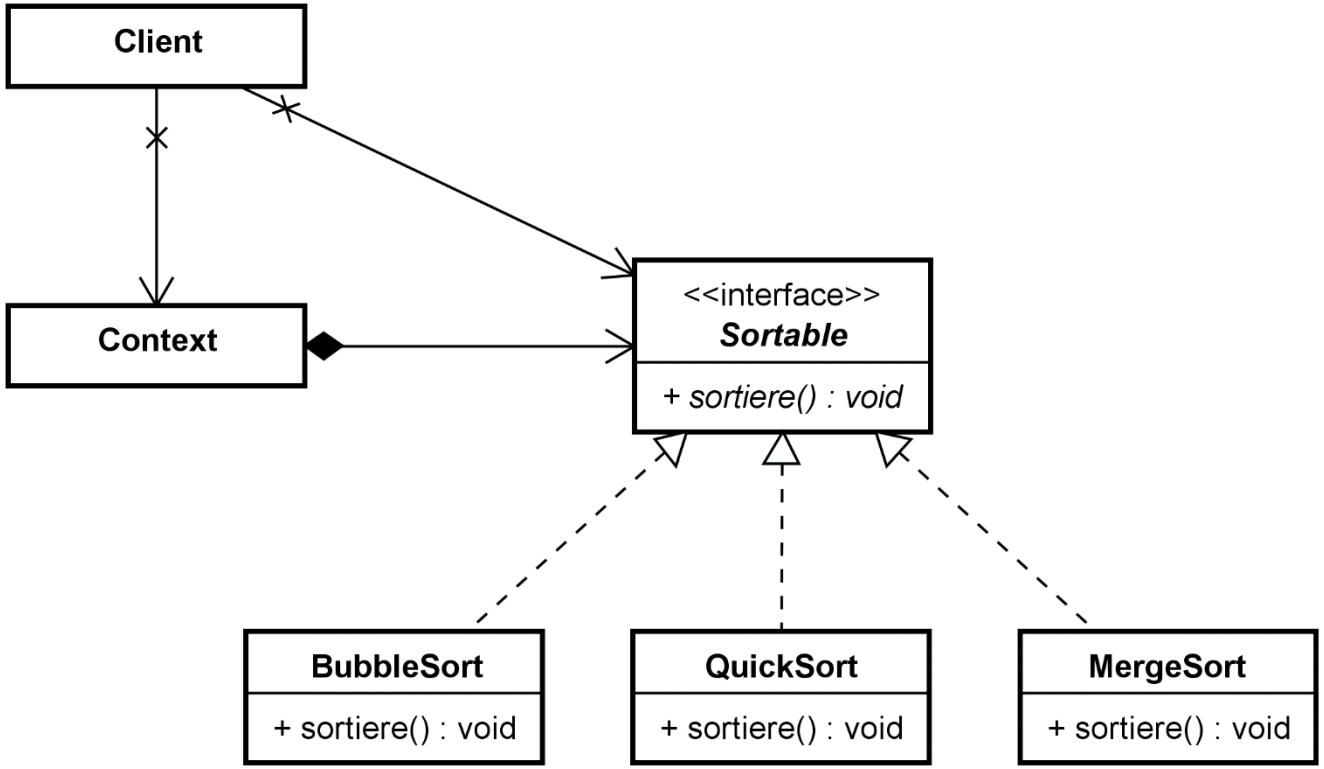
Beispiel 3: Strategie mit abstrakter Basisklasse

- Definiere eine Familie von Algorithmen, kapsle jeden Einzelnen und mache sie austauschbar. Somit ist es möglich den Algorithmus unabhängig vom ihn nutzenden Klienten zu variieren.

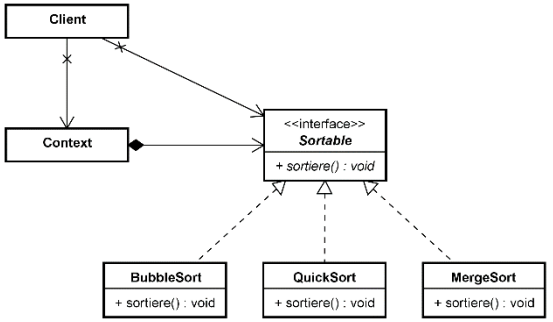


Beispiel 3: Strategie mit Interface

- Für Sprachen welche keine Mehrfachvererbung, stattdessen aber Interfaces anbieten (z.B. Java), ersetzt man die (voll-)abstrakte Basisklasse gerne durch ein Interface.

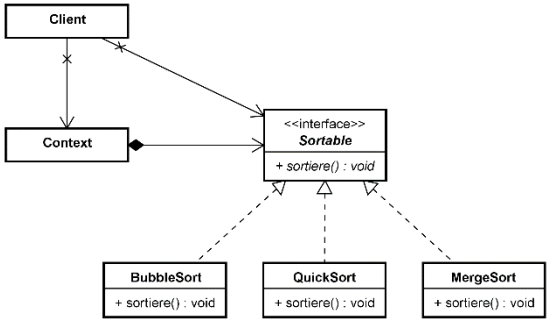


Beispiel 3: Strategie - Teilnehmer



- Strategie – Beispiel: **`Sortable`**
  - Vollabstrakte Klasse oder Interface, definiert die Schnittstelle.
- Kontext – Beispiel: **`Context`**
  - Ist Optional, kann auch direkt durch Client erledigt werden.
  - Besitzt eine Referenz auf die konkrete Strategie, erstellt diese ggf. auch gleich selber.
  - Stellt ggf. eine Datenschnittstelle für die Strategien zur Verfügung.
- Konkrete Strategien – Beispiel: **`BubbleSort`**, **`MergeSort`** etc.)
  - Implementieren einen konkreten Algorithmus.
  - Greifen evt. auf den Kontext zu (für die Daten).

Beispiel 3: Strategie



- Verhaltensmuster, objektbasiert
- Motivation für Einsatz:
  - Anbieten von unterschiedlichen Varianten/Implementationen von Algorithmen (z.B. Zeit vs. Speicher, Aufwand).
  - Eng verwandte Klassen, die sich nur im Verhalten unterscheiden, zusammenfassen.
  - Wenn der Bedarf nach unterschiedlichem Verhalten viele Bedingungsanweisungen zur Folge hätte.
- Empfehlung: Ein Pattern, das man leicht unterschätzt, und das sich auch bei sehr kleinen Methoden schon lohnen kann. Ausserdem lassen sich damit z.B. grosse und hässliche **`switch`**-Statements wunderbar eliminieren.

Empfehlungen zu Entwurfsmustern

Empfehlungen - Einsatz von Entwurfsmustern

- Voraussetzungen
  - Man muss die Entwurfsmuster kennen und verstehen!
  - Quellen: Literatur oder Internet.
- Sinnvolle Auswahl und überlegter Einsatz
  - Entwurfsmuster sind keine ultimative Lösung für Alles!
  - Erfahrung sammeln, Erfahrung notwendig.
  - Besser kein Muster einsetzen, als das Falsche!

Empfehlungen - Auswahl von Entwurfsmustern

- Es ist nicht immer einfach, das passende Muster (wenn überhaupt) auszuwählen!
- Sinnvolles Vorgehen:
  - Geht es um Erzeugung, Struktur oder Verhalten?
    - Passende Muster mit gleicher Aufgabe vorselektieren.
  - Welche Vor- und Nachteile bieten die Muster?
    - 'Bestes' Muster auswählen (am meisten Vorteile, grösste Vereinfachung etc.)
  - Wenn unentschieden: Wo haben Sie am meisten Freiheiten?
    - Muster mit grösster Flexibilität auswählen (grösstes Potential bei Erweiterungen / Wartung).

Empfehlungen - Verifikation des Entscheides

- Nachdem man sich für ein Muster entschieden hat, sollte man unbedingt anhand von realen oder fiktiven Beispielen **verifizieren**, ob die erhofften/erwünschten positiven Aspekte tatsächlich vorhanden sind!
- Beispiel anhand des Strategiemusters:
  - Sind weitere, sinnvolle Algorithmen in Form von Strategien implementierbar/vorstellbar?
  - Haben diese adäquaten Zugriff auf alle notwendigen Daten?
  - Lässt sich die konkrete Strategie sinnvoll bestimmen bzw. konfigurieren?
  - Wird das Resultat letztlich einfacher oder komplizierter?

Empfehlungen - Variieren von Entwurfsmustern

- Auch wenn Entwurfsmuster wohlüberlegt und vielfältig erprobt sind heisst das nicht, dass man sich stur daran halten muss!
- Entwurfsmuster sind 'nur' ein Konzept, welches auch wohlüberlegt verändert und optimiert werden darf.
  - Sturheit in der Implementation kann ein Design auch komplizierter oder aufwändiger machen.
  - Eingesetzte Sprache erlaubt evt. eine vereinfachte Umsetzung.
  - Im Gegenzug kann eine unbedachte Veränderung die spätere Entwicklung empfindlich behindern, oder gar die zentrale Idee eines Musters zerstören.
- Erfahrung und gesundes Augenmass notwendig!

  ➔ **Wer hat behauptet OO-Design sei einfach?** ☺

  Aber spannend auf jeden Fall!

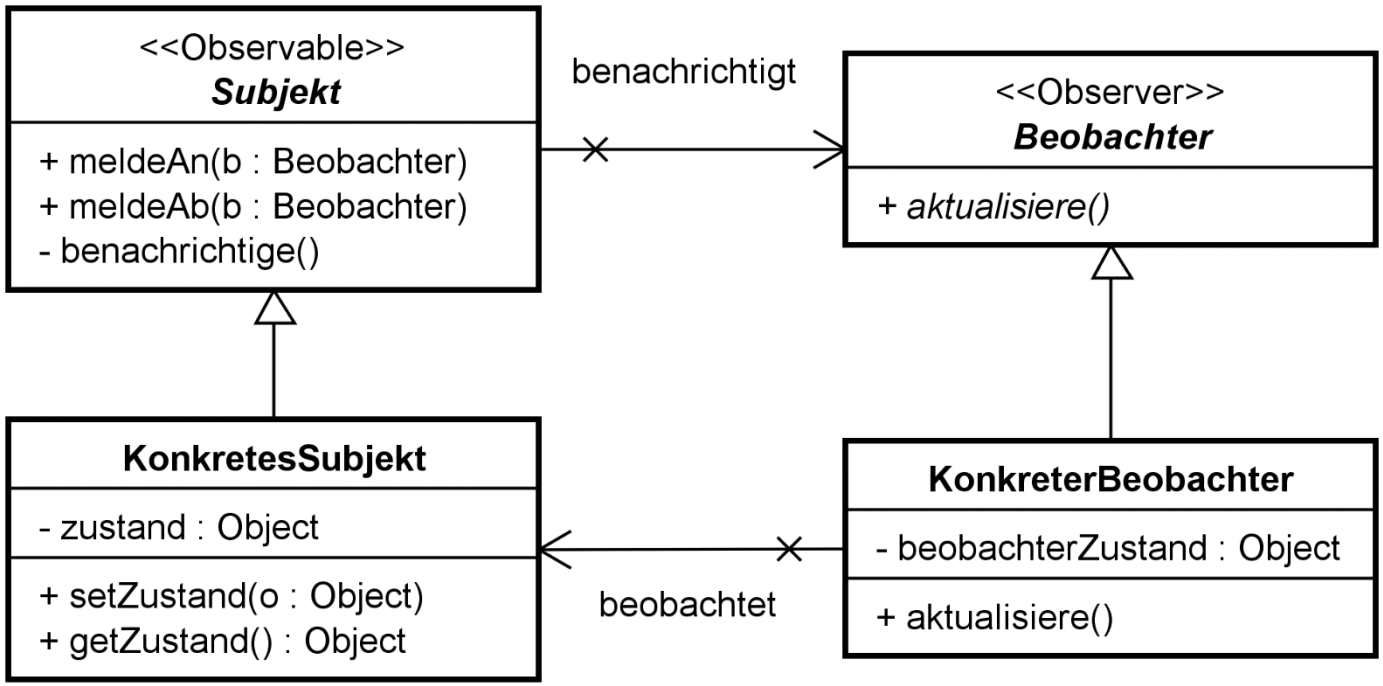
Empfehlungen - Kombination von Entwurfsmuster

- Passiert relativ häufig, wird in der Literatur aber kaum erwähnt.
- Eine wirklich effiziente Lösung lässt sich manchmal nur durch eine enge Kombination von Mustern erreichen.
  - Komplexität wird dadurch aber grösser.
  - Muster treten nicht mehr in 'Reinform' auf, und sind darum ggf. schwieriger als solche zu erkennen.
- Beispiele:
  - Fabrik für Zustände.
  - Fassade für Fabriken von dekorierten Strategien.

Beobachter

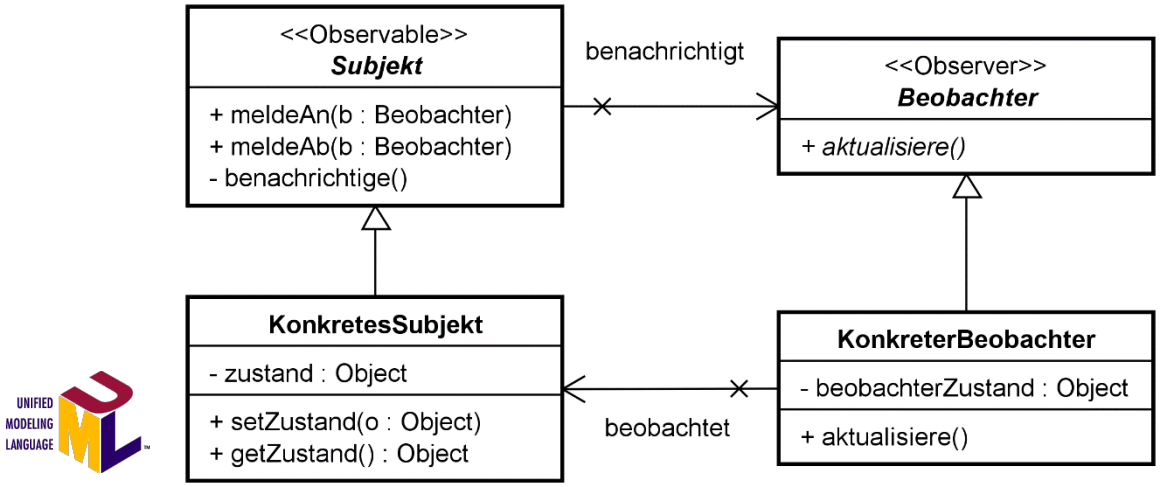
Beispiel 4: Beobachter

- Definiert eine Abhängigkeit zwischen einem Subjekt (**`Observable`**) dessen Zustand ändern kann, und einer Menge von Beobachtern (**`Observer`**) die darüber informiert werden sollen.



Beispiel 4: Beobachter - Teilnehmer

- Subjekt – **`Observable`**:
  - Verwaltet seine Beobachter (0..n).
  - Bietet Methoden zur An- und Abmeldung an.
- Beobachter – **`Observer`**:
  - Definiert eine Benachrichtigungsschnittstelle.
- Konkretes Subjekt / Konkreter Beobachter:
  - Konkrete Typen senden und empfangen Aktualisierungen.

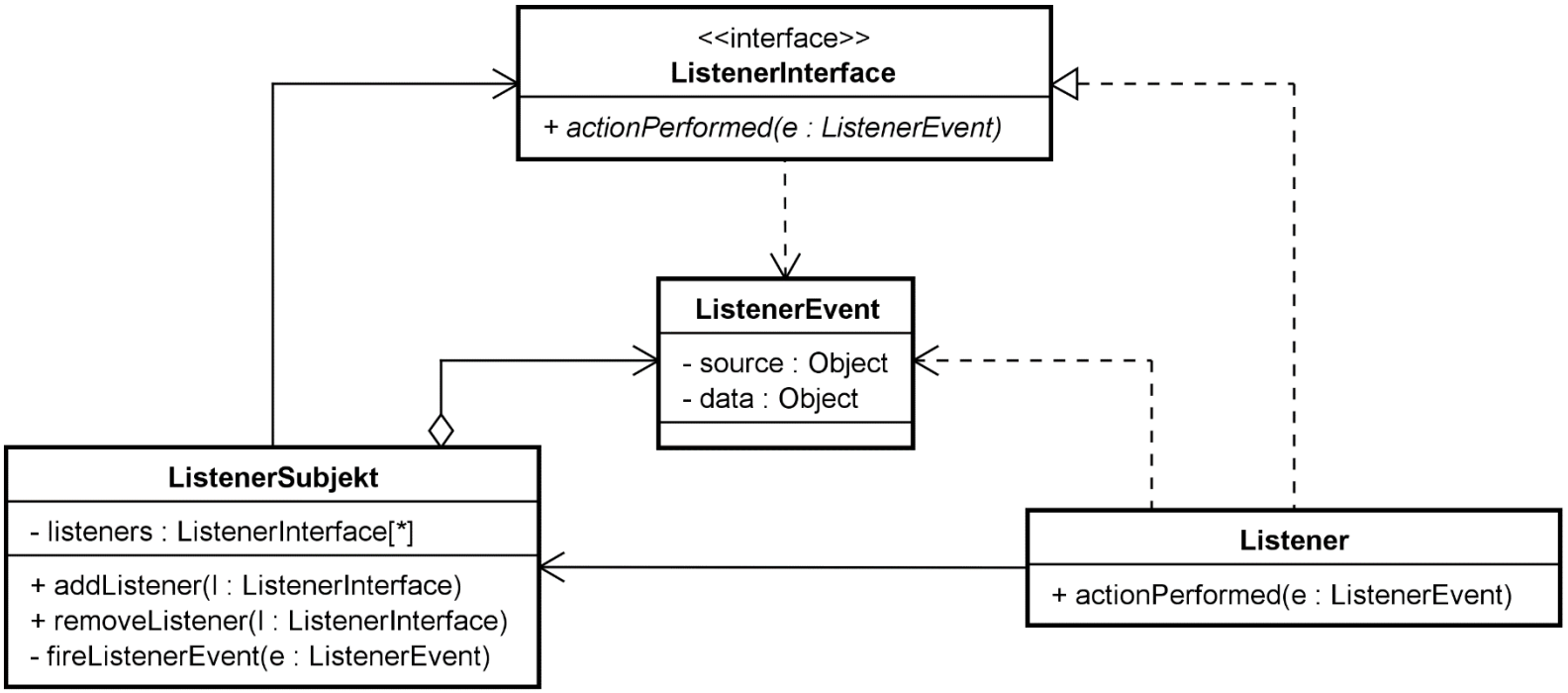


Beispiel 4: Beobachter

- Verhaltensmuster, objektbasiert.
- Motivation für Einsatz:
  - Wenn nur eine lose Kopplung der Zuhörer bestehen soll/darf.
  - Wenn die Anzahl der vorhandenen Zuhörer nicht interessiert.
  - Zur Kommunikation entgegen der Abhängigkeitsrichtung.
  - Auch zur Auflösung von zyklischen Referenzen.
- Sehr typisch für MVC: Änderungen des Modelles müssen an die verschiedenen Views propagiert werden.

Beispiel 4: Beobachter als `Event/Listener` (Java)

- Bei Java nutzt man als Ersatz für das Observer-Pattern das **`Event/Listener-Pattern`** welches auf Interfaces und Event-Klassen basiert.
  - Bekanntestes Listener-Interface: **`ActionListener`**



Beispiel 4: Namensgebung bei `Event/Listener` (Java)

- Event ➔ Eigentliches Subjekt.
- Eventquelle ➔ Verwaltet die Beobachter.
  - `public addXxxListener(...)`
  - `public removeXxxListener(...)`
  - `private fireXxxEvent(...)`
- Listener ➔ Beobachter
  - `public Xxx[Event|Performed](...)`
- Beispiel für `ActionListener`:
  - `addActionListener(ActionListener listener)`
  - `removeActionListener(ActionListener listener)`
  - `actionPerformed(ActionEvent actionEvent)`

Beispiel 4: Event/Listener-Modell in Java

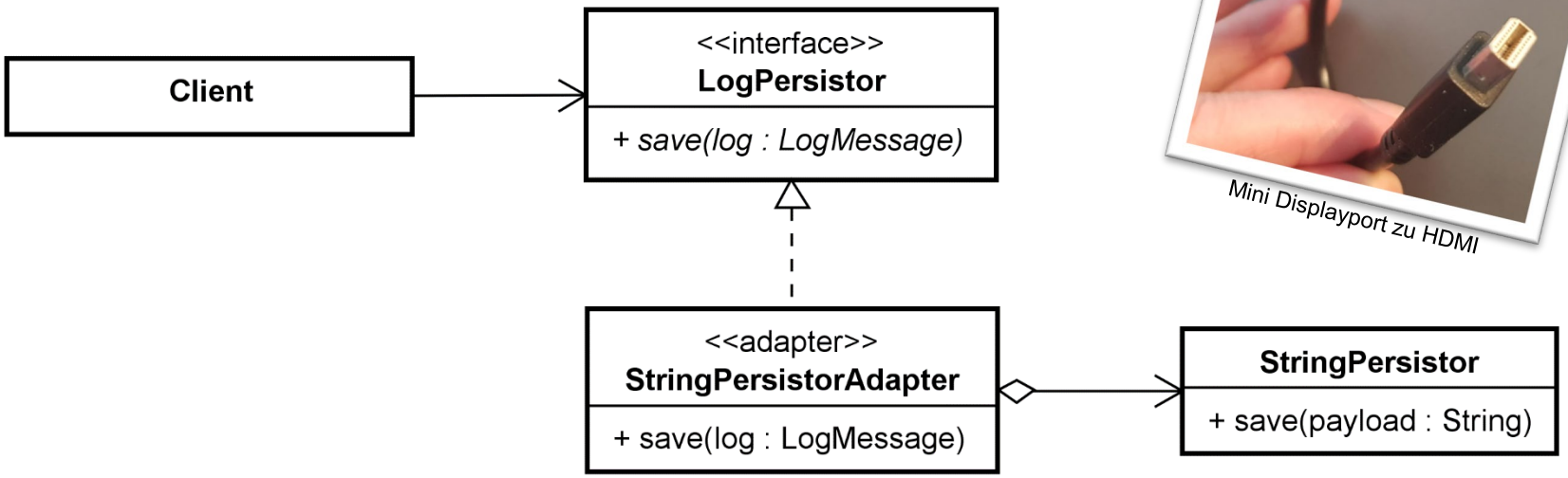
- Das Event/Listener-Modell von Java ist deutlich besser und flexibler als das «reine» Observer-Pattern der GoF!
  - Java kennt Interfaces, somit kann die Vererbung vom abstrakten Subjekt und Beobachter entfallen ➔ besseres Design!
- Java kennt seit Version 1.0 ein Interface **`Observer`** und eine Klasse **`Observable`**. Beide werden zur Verwendung schon lange **nicht mehr empfohlen**, und sind seit Java 9 (endlich) **deprecated**.

➔ **Bei Java konsequent das Event-/Listener-Modell verwenden!**

Adapter

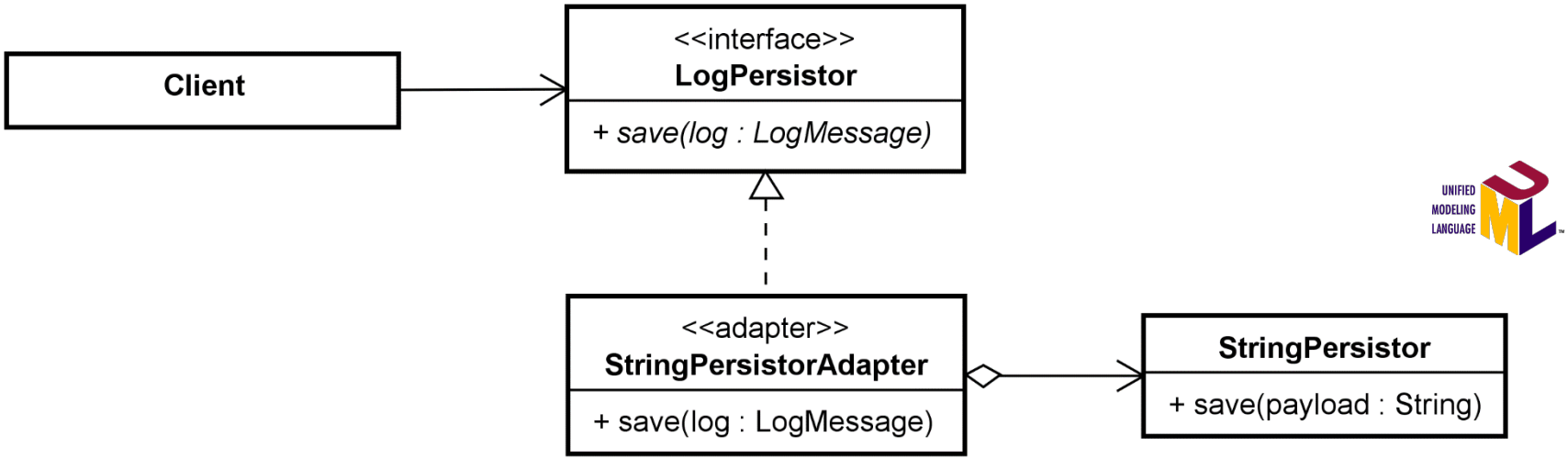
Beispiel 5: Adapter

- Anpassen der Schnittstelle einer Klasse an die von den Klienten erwartete (Ziel-)Schnittstelle.
- Auch bekannt als „Wrapper"-Pattern.



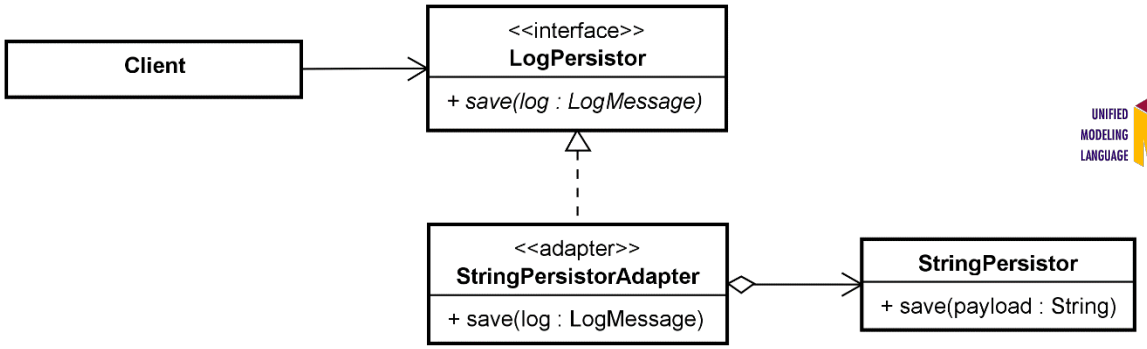
Beispiel 5: Adapter

- Strukturmuster, klassen- oder objektbasiert
- Motivation für Einsatz:
  - Einfachere Wiederverwendung von existierenden Klassen oder Komponenten, deren Schnittstelle aber unpassend ist.
  - Eine möglichst allgemeine Schnittstelle zu implementieren, und diese dann prinzipiell durch Adapter anzupassen.



Beispiel 5: Adapter - Teilnehmer

- Interface – Beispiel: **`LogPersistor`**
  - Die effektiv gewünschte Zielschnittstelle.
  - Kann eine abstrakte Klasse oder ein Interface sein.
- Adapter – Beispiel: **`StringPersistorAdapter`**
  - Verwendet die adaptierte Klasse/Objekt.
  - Spezialisiert oder implementiert die Zielschnittstelle.
- Adaptierte Klasse – Beispiel: **`StringPersistor`**
  - Adaptierte Klasse, deren Schnittstelle adaptiert (vgl. wrappen) werden soll.



Ergänzende Hinweise

Kommentar zu den ausgewählten Patterns

- Einfache Patterns, die aber sehr häufig eingesetzt werden!
- Mögliche Einstiegspunkte in die Anwendung von Entwurfsmustern:
  - Denken Sie an Ihre eigenen Projekte!
  - Gibt es Muster, die Sie in Ihren Projekten einsetzen könnten?
- Am Anfang müssen Sie auch aus Fehler lernen.
  - Gefahr: Mit Kanonen auf Spatzen schiessen.
  - Konkret: Das Muster kann plötzlich komplexer als das darin enthaltene/gelöste Fachproblem sein.

➔ **Entwurfsmuster sind kein goldener Hammer!**

Logger-Projekt - Aufträge

- Beachten Sie die Muss-Features im Projektauftrag!
  - Implementation eines Adapter-Pattern.
  - Implementation eines Strategy-Pattern.
- Gute Beispiele für «**S**eparation **o**f **C**oncerns» (SoC)
  - Lassen sich dadurch sehr gut Unit Testen.
  - Führen zu einem feingranularem Design (kleine[re] Klassen).
  - Einfaches Testing mit ➔Test Doubles möglich.

➔Es sind somit explizit **keine** «Schulbeispiele»!
  - Nur als Muss-Features ist es aussergewöhnlich. ☺

Zusammenfassung

- Relativ grosse Zahl an Entwurfsmuster verfügbar (20+)
  - Für verschiedene Zwecke / Problemlösungen nutzbar.
  - Weitgehend sprachenunabhängiger Entwurf.
  - Gut und ausführlich dokumentiert.
  - Rad muss nicht immer wieder neu erfunden werden.
  - Hoher Bekanntheitsgrad und Wiedererkennung.
- Nicht für Alles gibt es ein Entwurfsmuster das passt
  - Entwurfsmuster sind kein goldener Hammer!
  - Auswahl des geeigneten Musters nicht immer einfach.
  - Erzwungener Einsatz geht meistens schief.
  - Manchmal Kombinationen von Mustern anwenden.

Quellen 1 - Literatur

- *Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides:*

  Entwurfsmuster

  MITP

  Januar 2015

  ISBN: 978-3-8266-9700-5

Original

- *Eric Freeman, Elisabeth Freeman, Kathy Sierra, Bert Bates:*

  Entwurfsmuster von Kopf bis Fuss

  O'Reilly Deutsch

  Februar 2015

  ISBN: 978-3-95561-986-2

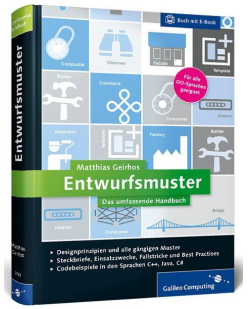
- *Matthias Geirhos:*

  Entwurfsmuster

  Rheinwerk Computing

  Mai 2015

  ISBN: 978-3-8362-2762-9



Quellen 2

- Internet (Auswahl):
  - Design Patterns (Wikipedia, englisch)
    https://en.wikipedia.org/wiki/Design_Patterns
  - Entwurfsmuster (Wikipedia, deutsch)
    https://de.wikipedia.org/wiki/Entwurfsmuster
  - Patterns, Anti-Patterns, Refactoring und UML
    https://sourcemaking.com/design_patterns
  - Entwurfsmuster - Übersicht (Uni Rostock)
    https://wwwswt.informatik.uni-rostock.de/deutsch/Infothek/Entwurfsmuster/patterns
  - Refactoring Guru – Design Patterns
    https://refactoring.guru/design-patterns

Fragen?