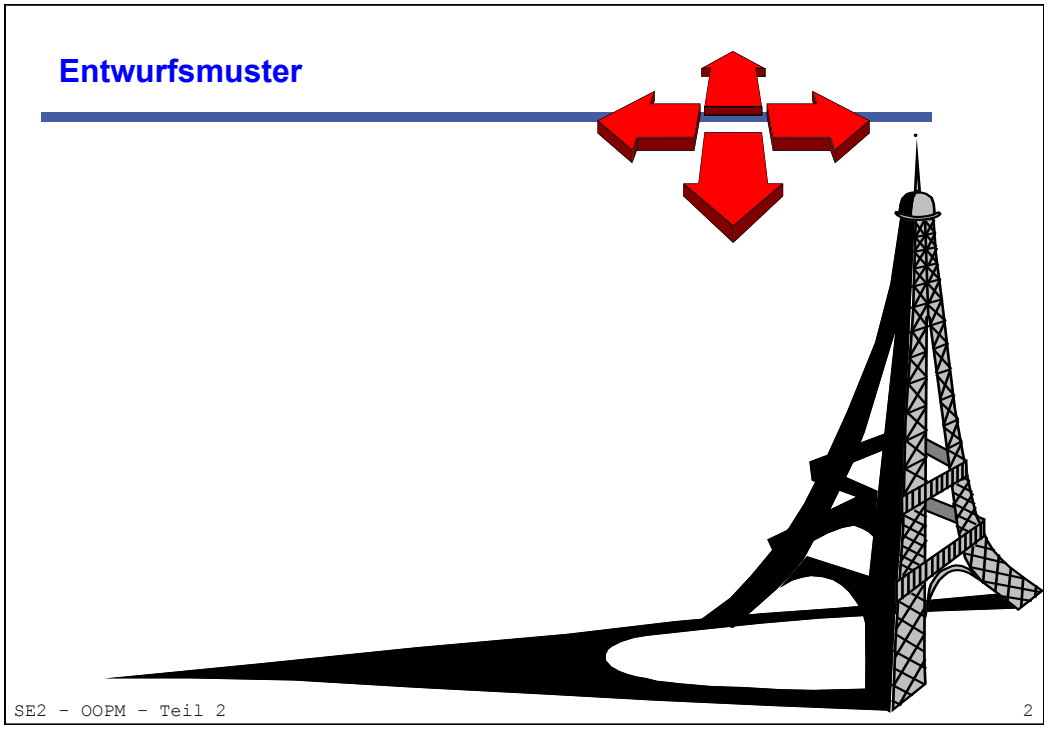


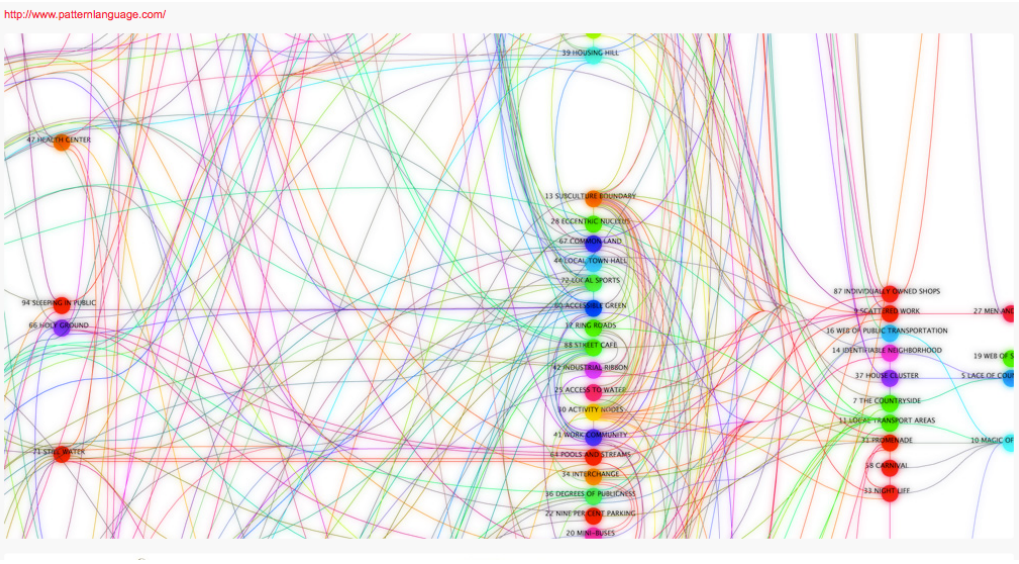
Mikroarchitekturen: Muster in der Architektur

Christopher Alexander:

"Each pattern describes a **problem** which occurs over and over again in our environment, and then describes the **core of the solution** to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."
[Alexander, Ishikawa, Silverstein. A pattern language. Oxford University Press, 1977]

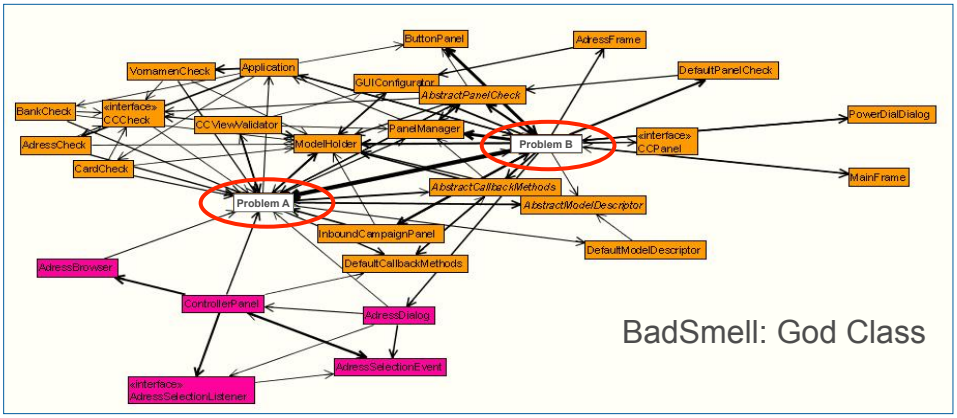
"Each pattern is a three part rule, which expresses a relation between a certain **context**, a **problem**, and a **solution**."
[Alexander. The timeless way of building. Oxford University Press, 1979]

Die meisten Autoren im Bereich Design Patterns schließen sich Alexander an (z.B. Gamma et al., Beck, Johnson, Schmidt, Buschmann, Coplien).



Chaos auf Klassenebene

- 30% der Klassen sind in Klassenzyklen
- 60% der Packages sind in Zyklen



- 98% der Klassen haben zu weniger als 10 Klassen Beziehungen
- „Problem A“ kennt 21 Klassen und wird von 49 benutzt
- „Problem B“ kennt 63 Klassen und wird von 18 Klassen benutzt

Muster

Muster(engl. **pattern**)

Ein Muster ist eine Abstraktion von einer konkreten Form, die wiederholt in bestimmten, nicht willkürlichen Kontexten auftritt.

- Der Musterbegriff ist hier sehr allgemein. Er ist weder auf Softwareentwicklung noch auf eine bestimmte Verwendung von Mustern zugeschnitten.
- Die (bekannte) Definition **"A pattern is a solution to a recurring problem in a context"** ist auf die Lösung von Entwurfsproblemen ausgerichtet.

Begriffsdefinition: Software-Architektur

- Man versteht unter Software-Architektur die Einteilung eines Systems in Elemente, deren Schnittstellen, die Prozesse und Abhängigkeiten zwischen ihnen, sowie die benötigten Ressourcen [McDermid 91].
- Als Elemente sind dabei die Makrostrukturen (wie Packages, Frameworks oder Schichten) gemeint, die eine überschaubare und handhabbare Organisation der Systeme erlauben.

- Entwickler großer Softwaresysteme benötigen ein explizit vorliegendes Modell der Softwarearchitektur, wenn sie das System verstehen und systematisch weiterentwickeln wollen.
  Ein explizites Modell kann z.B. als Klassen- und Komponentendiagramm in UML vorliegen.

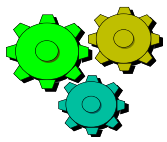
Mikroarchitekturen in objektorientierter Software

Ein **Entwurfsmuster** (engl.: **design pattern**)

- beschreibt abstrakt eine bewährte Lösung für ein bestimmtes und häufig wiederkehrendes Problem des **objektorientierten Softwareentwurfs**;
- **entsteht durch die Analyse und Überarbeitung vorhandener Designlösungen, setzt also Entwurfs- und Programmiererfahrung voraus**;
- kann in seiner **Struktur** verstanden werden als eine Menge von Klassen, die festgelegte Verantwortlichkeiten haben und in einer definierten Vererbungs- bzw. Benutzungsbeziehung zueinander stehen;
- kann in seiner **Dynamik** verstanden werden als eine Menge von Objekten, die nach einem beschreibbaren Prinzip interagieren bzw. erzeugt werden ;
- kann immer nur zusammen mit dem Entwurfsproblem beschrieben werden, das es lösen soll.

Wir entdecken ein Entwurfsmuster

„Definiere eine 1-zu-n Beziehung zwischen Objekten, so dass die Änderung des Zustands eines Objekts dazu führt, dass alle abhängigen Objekte benachrichtigt und automatisch aktualisiert werden.“



Ein „Ueberweiser“

```
public class Ueberweiser {
    public void setzeAuftraggeber( int ktoNr, int blz ) {...}
        -- setzt das Konto des Auftraggebers
    public void setzeEmpfaenger( int ktoNr, int blz ) {...}
        -- setzt das Konto des Empfaengers
    public void ueberweisen( float betrag )
        -- überweist 'betrag' vom 'AuftraggeberKonto' auf das 'EmpfaengerKonto'
    public float empfaengerSaldo() {...}
        -- liefert den aktuellen Kontostand des Empfaengers
    public float auftraggeberSaldo {...}
        -- liefert den aktuellen Kontostand des Auftraggebers
    public boolean gueltigesKonto( int ktonr, int blz) {...}
        -- prüft, ob ein Konto mit 'ktonr' bei der Bank 'blz'existiert
    public boolean istUeberweisungMoeglich() {...}
        -- prüft, ob die Konten von Auftraggeber und Empfaenger bereits bestimmt worden sind
    public boolean istAuftraggeberSaldoVeraendert () {...}
        -- liefert 'true', wenn das Saldo des Auftraggebers verändert wurde
    public boolean istEmpfaengerSaldoVeraendert () {...}
        -- liefert 'true', wenn das Saldo des Empfängers verändert wurde
}
```


Operationen an der Schnittstelle des Ueberweisers

Wir unterscheiden:

- **verändernde Operationen**, die eine Veränderung des Zustands bewirken setzeAuftraggeber, setzeEmpfaenger, ueberweisen,
- **sondierende fachliche Operationen**, mit denen Informationen erfragt werden können empfaengerSaldo, auftraggeberSaldo,
- **sondierende boolesche Operationen** mit denen u.a. Parameterwerte und Reihenfolgebedingungen getestet werden können istEmpfaengerSaldoVeraendert, istAuftraggeberSaldoVeraendert, gueltigesKonto, istUeberweisungMoeglich.

Die Schnittstelle des Ueberweisers macht keine Annahmen über die Art ihrer Benutzung. (Graphikkomponente? Eine andere Klasse?)

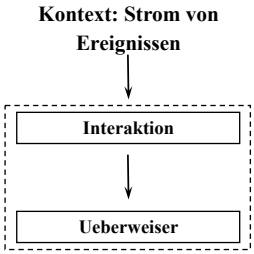
Wir wollen den Ueberweiser interaktiv benutzen

Zu lösende Aufgaben:

- Entgegennahme eines Stroms von Ereignissen, die durch Aktionen des Benutzers ausgelöst werden,
- Abstraktion vom zugrundeliegenden Fenstersystem unter dessen Verwendung Ein- und Ausgabe erfolgt (z.B. mit Knöpfen, Menüs).

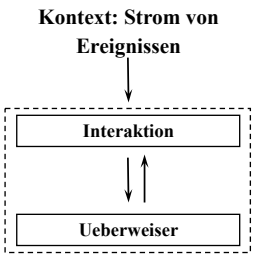
Grundproblem der Konstruktion: Zwei Klassen müssen sich kennen.

- **Eine Aktion des Benutzers löst eine Reaktion des Programms aus.**
- Das bedeutet technisch: die Interaktion benutzt den Ueberweiser.



Grundproblem der Konstruktion: Zwei Klassen müssen sich kennen.

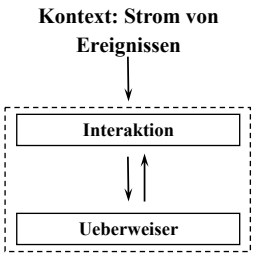
- **Eine Aktion des Benutzers löst eine Reaktion des Programms aus.**
- Das bedeutet technisch: die Interaktion benutzt den Ueberweiser.
- **Der Ueberweiser soll seinen Zustand an der Oberfläche zeigen.**
- Das bedeutet technisch: Der Ueberweiser benutzt die Interaktion.



Grundproblem der Konstruktion: Zwei Klassen müssen sich kennen.

- **Eine Aktion des Benutzers löst eine Reaktion des Programms aus.**
- Das bedeutet technisch: die Interaktion benutzt den Ueberweiser.
- **Der Ueberweiser soll seinen Zustand an der Oberfläche zeigen.**
- Das bedeutet technisch: Der Ueberweiser benutzt die Interaktion.

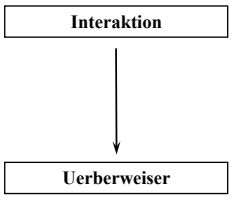
- **Sich zyklisch benutzende Klassen müssen wir vermeiden.**
- **Wir erkennen das Grundproblem unserer Konstruktion: Das Rückkopplungsproblem**



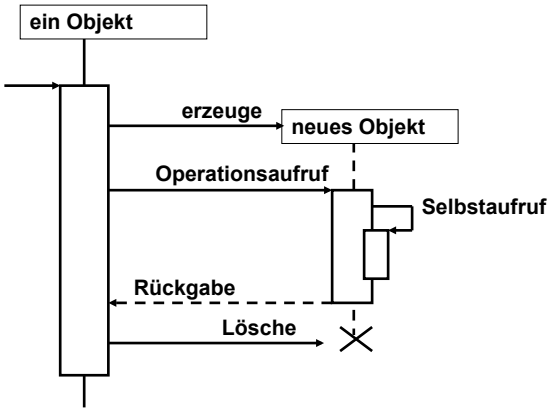
Lösungsansätze für das Rückkopplungsproblem (1)

Die Interaktion kennt die Effekte von Operationsaufrufen und ruft geeignete **sondierende Operationen**, um einen veränderten Zustand des Ueberweisers darzustellen.

Das Geheimnisprinzip ist verletzt, da die Interaktion Kenntnis über die Umsetzung von Aufrufen besitzt !



Einschub: UML-Sequenzdiagramme



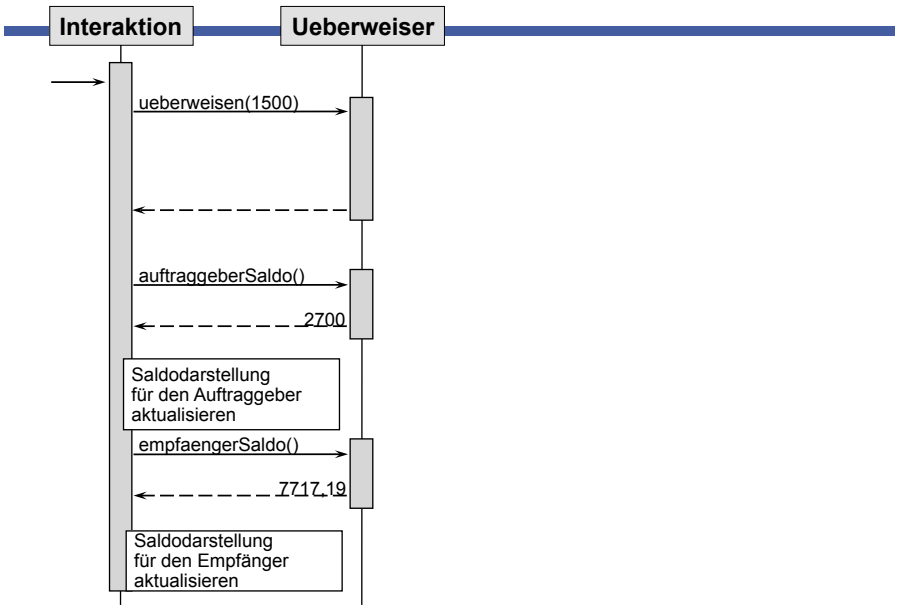
- Objektinteraktionen in zeitlicher Reihenfolge.
- Die an einem Szenario beteiligten Objekte und Klassen und die Operationsaufrufe die zwischen ihnen ausgeführt werden, um die Funktionalität eines UseCases zu erfüllen.
- Pragmatik (in UML): Soll während der Analysephase eingesetzt werden. Außerdem: “Keep it simple, stupid.”

Einschub: UML-Sequenzdiagramme

- Ein Sequenzdiagramm besitzt zwei Dimensionen
  - vertikal: Zeit
  - horizontal: beteiligte Objekte
- Der Zeitablauf erfolgt von oben nach unten
  - normalerweise ist nur die Reihenfolge der Nachrichten signifikant
  - in Echtzeitanwendungen kann die Zeitachse auch eine Metrik besitzen
- Die horizontale Ordnung der Objekte ist nicht signifikant.

- **Ein Sequenzdiagramm ist die konkretisierte Darstellung eines Szenarios:**
  - Beschreibt die im Szenario auftretenden Ereignisse in ihrer zeitlichen Abfolge.
  - Benennt die am Szenario beteiligten Objekte.
- **Ein Sequenzdiagramm zeigt:**
  - den Nachrichtenaustausch zwischen den Objekten
  - die Lebenszeit der Objekte
  - die Aktivitätszeiten der Objekte

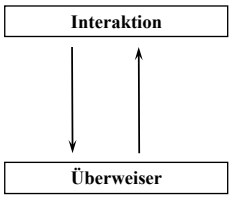
Lösungsansätze für das Rückkopplungsproblem (1)



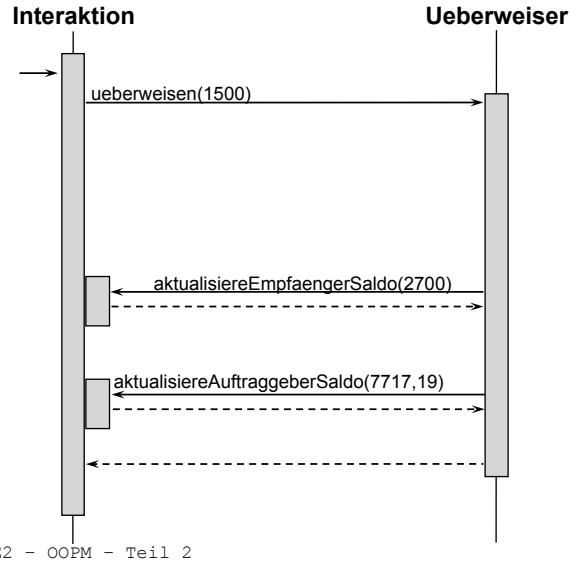
Lösungsansätze für das Rückkopplungsproblem (2)

Am Ende eines **Operationsaufrufs** ruft der Ueberweiser seinerseits die Interaktion, um einen veränderten Zustand aktualisiert darstellen zu lassen.

Die Aufgabentrennung von Ueberweiser und Interaktion ist verletzt, da der Ueberweiser über die Realisierung der Darstellung Kenntnis besitzt !



Lösungsansätze für das Rückkopplungsproblem (2)

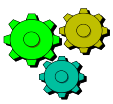


Lösungsansatz für das Rückkopplungsproblem (3)

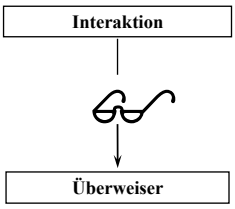
Nach jedem Systemereignis ruft die Interaktion die überprüfenden Funktionen des Ueberweisers, um einen veränderten Zustand unmittelbar nachzuvollziehen.

Wir bezeichnen diesen Lösungsansatz als **Polling-Ansatz**.

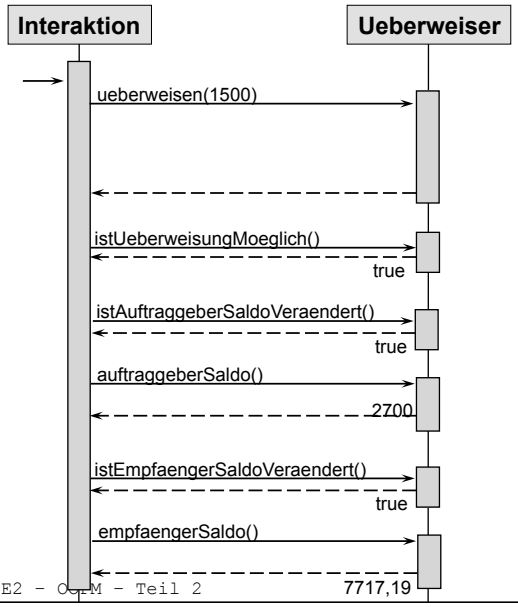
Die Interaktion ist fast ausschließlich mit Polling beschäftigt !



Softwaretechnisch ist **Polling** die sauberste der drei Lösungen und vermeidet die Nachteile der ersten beiden Lösungen.

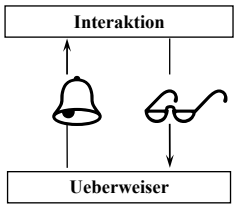


Lösungsansätze für das Rückkopplungsproblem (3)



Weiterentwicklung des Polling-Ansatzes: Der benachrichtigte Beobachter

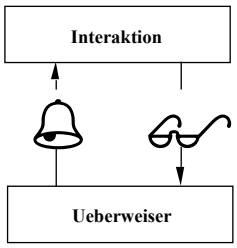
•Konzept:
- Der Ueberweiser signalisiert Zustandsänderungen.
- Die Interaktion beachtet diese Signale und ruft entsprechende **sondierende Operationen**.
- Folge: Die Interaktion muss im Ueberweiser bekannt sein.



Aufrufe, Ereignisse, Signale

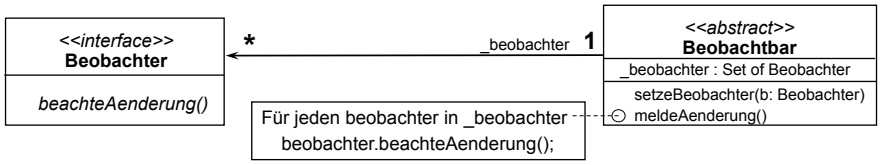
Wir unterscheiden:

- Beim **Aufruf** (Call) kennt der Rufer (Klient) den Gerufenen (Anbieter) und erwartet eine bestimmte Dienstleistung.
- Ein **Ereignis** (Event) wird vom Erzeuger oder Verteiler an einen bestimmten Empfänger weitergeleitet. Eine Reaktion des Empfängers wird nicht erwartet.
- Ein **Signal** (Broadcast) wird vom Erzeuger an die Umgebung ausgesandt. Prinzipiell sind die Empfänger des Signals anonym. Die Reaktion auf ein Signal ist meist ein Aufruf des Erzeugers.



Da uns die gängigen oo Sprachen keinen eigenen Ereignis- oder Signalmechanismus anbieten, konstruieren wir die Benachrichtigung mit Hilfe des Aufrufmechanismus.

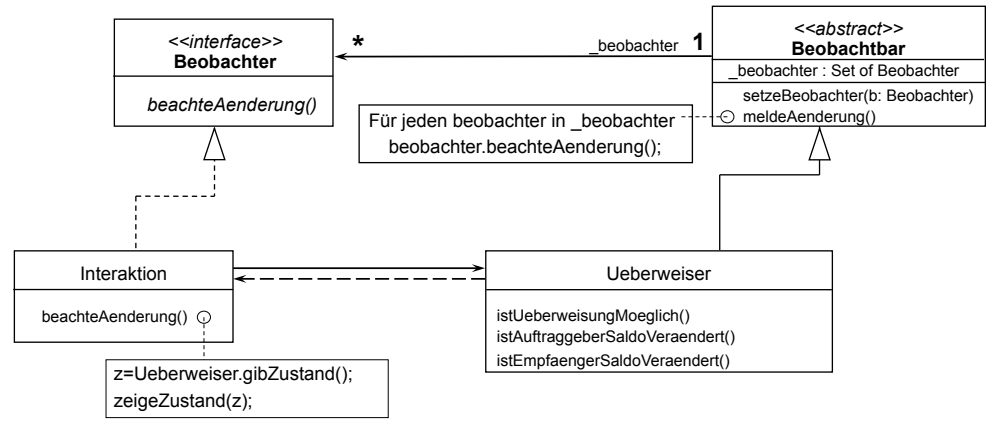
Beobachten und beobachtet werden: Beobachter und Beobachtbar



Beteiligte:

- Das **Interface Beobachter**: Jede Klasse, die eine andere Klasse beobachten können möchte, muss dieses Interface implementieren. Exemplare implementierender Klassen registrieren sich bei einem beobachtbaren Exemplar durch einen Aufruf von setzeBeobachter(). In beachteAenderung() wird eine Reaktion auf das Änderungssignal (der beobachtbaren Klasse) implementiert.
- Die **Klasse Beobachtbar**: Eine Klasse, die beobachtbar sein will, erbt von dieser Klasse. Registrierte Beobachter werden durch einen Aufruf von meldeAenderung() über eine Zustandsänderung benachrichtigt.

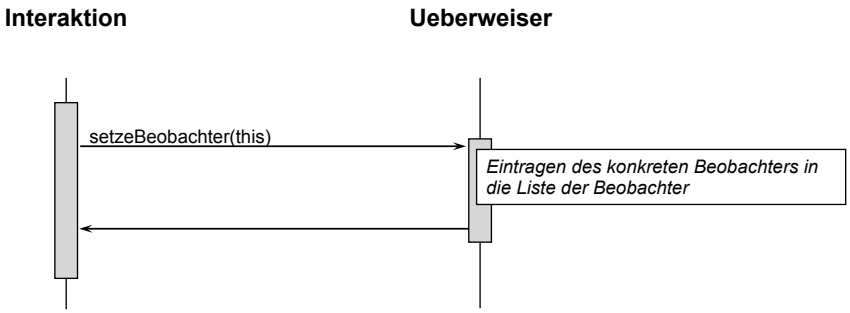
Beobachter und Beobachtbar im Kontext des Überweisers



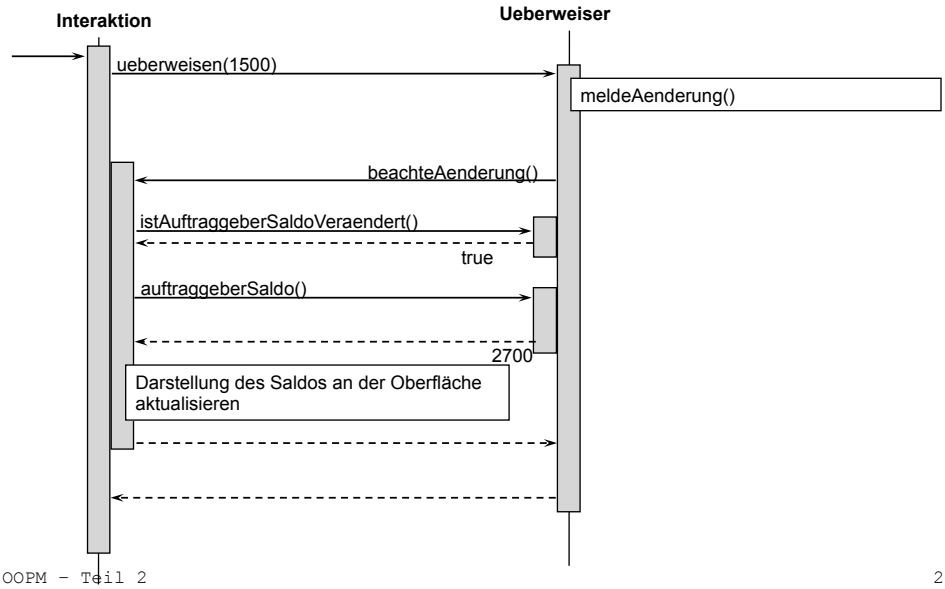
Eine Interaktion kennt den Ueberweiser unter der vollen Schnittstelle.

Der Ueberweiser benachrichtigt die Interaktion unter ihrer abstrakten Schnittstelle Beobachter.

Dynamik: Registrieren eines Beobachters



Dynamik des Beobachter-Musters



Das Beobachter-Muster

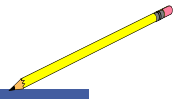
Zweck

Das Beobachter-Muster wird zum Synchronisieren von Objektänderungen verwendet. „Definiere eine 1-zu-n-Abhängigkeit zwischen Objekten, so dass die Änderung des Zustandes eines Objekts dazu führt, dass alle abhängigen Objekte benachrichtigt und automatisch aktualisiert werden.“ [GHJV, S. 11]

Anwendbarkeit/Motivation (http://www.die.informatik.uni-siegen.de/lehre/EI/informatik1/vorlesung_material/21_MVC.pdf)

- Konsistenz (in sich stimmig, keine Widersprüche) der Objekte sicherstellen
- Klassen nicht miteinander koppeln → Wiederverwendbarkeit
- Vorgehensweise: publiziere und abonniere (publish − subscribe)

Beschreibung von Entwurfsmustern



Eine **Musterbeschreibung** besteht meist aus den folgenden Teilen:

- dem **Namen** des Entwurfsmusters,
- dem **Problem**, das mit Hilfe des Musters gelöst werden soll,
- der **Kontext**, in dem sich das Problem stellt,
- der **Lösung**, mit der die Organisation von Klassen in einer Klassenhierarchie und die Gestaltung ihrer Interaktion vorgegeben werden,
- die (positiven und negativen) **Konsequenzen** der Musteranwendung.

Von Gamma et al. wird eine feiner ausgearbeitete Notation verwendet:

- Name
- Ziel
- Motivation
- Struktur
- Teilnehmer
- Zusammenarbeit
- Implementation
- Anwendbarkeit
- Konsequenzen

Unterteilung von Entwurfsmustern

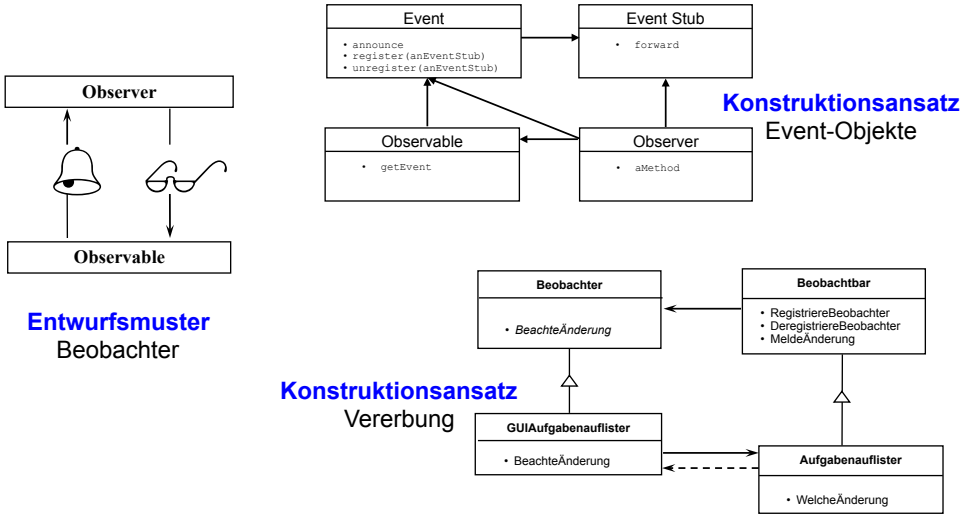
Der **allgemeine Teil eines Entwurfsmusters**:

- beschreibt die abstrakten Modellierungselemente und Bezüge für den softwaretechnischen Entwurf,
- beschreibt die zusammenarbeitenden Objekte und Klassen, die maßgeschneidert sind, um ein allgemeines Entwurfsproblem in einem bestimmten Kontext zu lösen,
- muss durch unterschiedliche Konstruktionsansätze konkretisiert werden.

Ein **Konstruktionsansatz** als der konstruktive Teil eines Entwurfsmusters:

- beschreibt die konkreten Elemente und Bezüge des softwaretechnischen Entwurfs,
- ist i.d.R. eine Implementationsvariante, die einen Aspekt des Musters betont und auf eine bestimmte Programmiersprache zugeschnitten ist.
- kann unmittelbar in eine programmiersprachliche Implementation umgesetzt werden.

Zusammenhang Entwurfsmuster und Konstruktionsansatz



Zum Nutzen von Entwurfsmustern

Entwurfsmuster

- helfen, existierende Softwareentwürfe zu analysieren und zu reorganisieren;
- erleichtern die Einarbeitung in Software-Architekturen (z.B. Klassenbibliotheken, Rahmenwerke), solange sie auf der Basis von bekannten Entwurfsmustern dokumentiert sind;
- sind “Mikroarchitekturen”, die sich von erfahrenen Entwicklern als Bausteine innerhalb größerer Software-Architekturen wiederverwenden lassen (Wiederverwendung von Design-Lösungen statt Wiederverwendung von Code);
- stellen uns die Elemente einer Sprache, in der wir über Software-Architekturen nachdenken und kommunizieren können;
- sollen die softwaretechnische Qualität von Entwürfen erhöhen (z.B. ihre Wiederverwendbarkeit und Erweiterbarkeit).

Klassifikationsvorschlag der GoF für ihre Muster

		Aufgabe		
		Erzeugungsmuster	**Strukturmuster**	**Verhaltensmuster**
Gültigkeits-bereich	**Klassen-basiert**	Objekterzeugung wird in Unterklassen verlagert	Vererbung wird genutzt, um Klassen zusammenzuführen	Vererbung wird verwendet, um Algorithmen und Kontrollfluß zu beschreiben
	Objekt-basiert	Objekterzeugung wird an ein anderes Objekt delegiert	Möglichkeiten Objekte zusammen zu führen	Objekte arbeiten zusammen, um eine Aufgabe auszuführen, die ein einzelnes Objekt nicht in der Lage wäre zu erfüllen

SE2 – OOPM – Teil 2 35

GoF: 23 Entwurfsmuster klassifiziert

		Aufgabe		
		Erzeugungsmuster	**Strukturmuster**	**Verhaltensmuster**
Gültigkeits-bereich	**Klassen-basiert**	Fabrikmethode	Adapter (klassenbasiert)	Interpreter Schablonenmethode
	Objekt-basiert	Abstrakte Fabrik Erbauer Prototyp Singleton	Adapter (objekt-basiert) Brücke Dekorierer Fassade Fliegengewicht Kompositum Proxy	Befehl Beobachter Besucher Iterator Memento Strategie Vermittler Zustand Zuständigkeitskette

Zusammensetzung von Klassen und Objekten

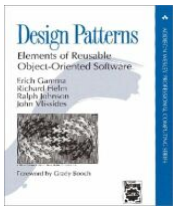
Zusammenarbeit von Klassen und Objekten

SE2 – OOPM – Teil 2 36

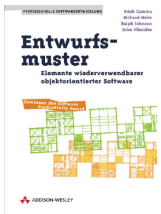
Ein Klassiker der Softwaretechnik

„Design Patterns: Elements of ReusableObject-Oriented Software“

der Gang of Four, mit 23 Entwurfsmustern.

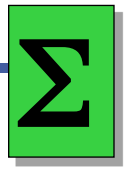


1995



1996

Zusammenfassung Entwurfsmuster



- Entwurfsmuster sind ein wesentliches Hilfsmittel des objektorientierten Software-Entwurfs.
- Zum softwaretechnischen Handwerkszeug gehört die Kenntnis über:
  - Erzeugungs-
  - Struktur- und
  - Verhaltensmuster
- Entwurfsmuster sind das zentrale konzeptionelle Bindeglied zwischen
  - den Mitteln einer objektorientierten Programmiersprache und
  - Überlegungen zur Architektur großer Softwaresysteme.