Design Pattern Teil 2

- Ein anderes Beispiel
  - Anwendung
- Ein Pattern-Katalog
- Pattern-Beschreibung

DHBW Duale Hochschule Baden-Württemberg Mannheim

Zur Erinnerung: Muster

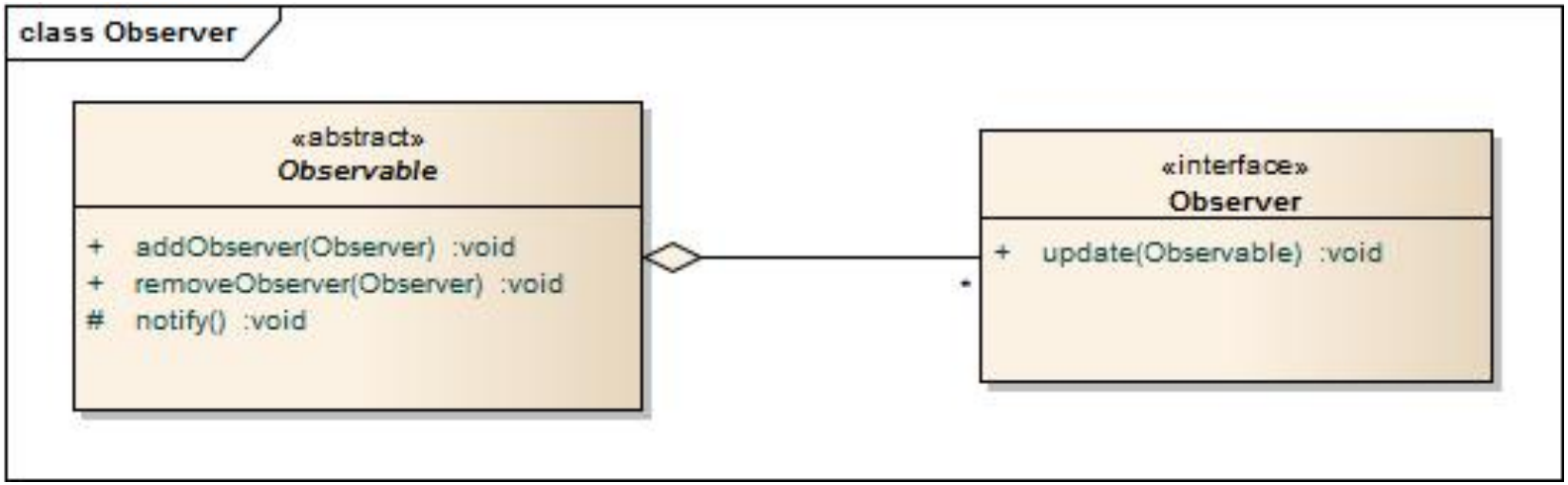
- Problembeschreibung + Kern der Problemlösung + Konsequenzen (pos. + neg.)
- Keine genaue Spezifikation (möglich)
- Die Lösung kann in verschiedenen Umgebungen wieder verwendet werden

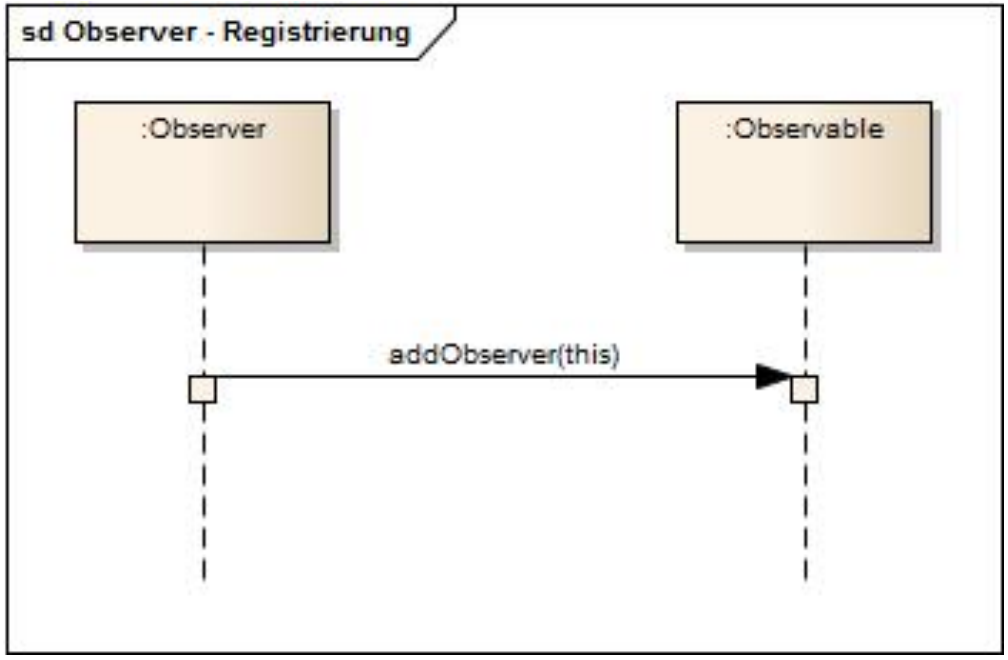
Wir haben bereits das Singleton betrachtet

- Haben Sie Fragen dazu?

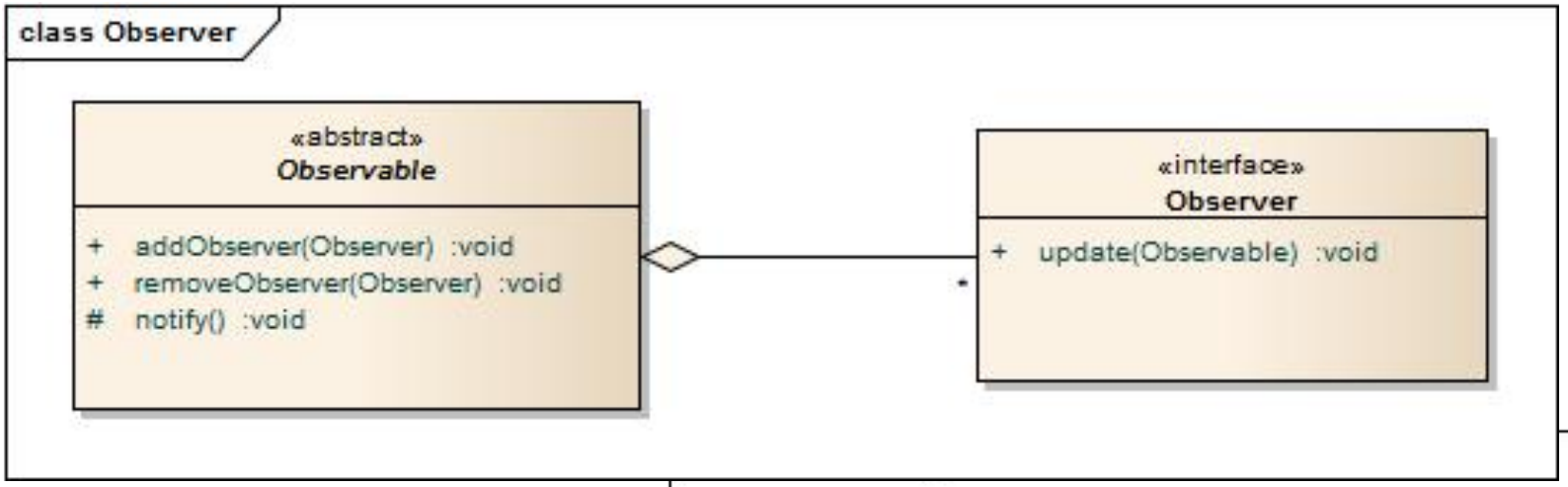
Wir schauen uns ein weiteres Beispiel an...

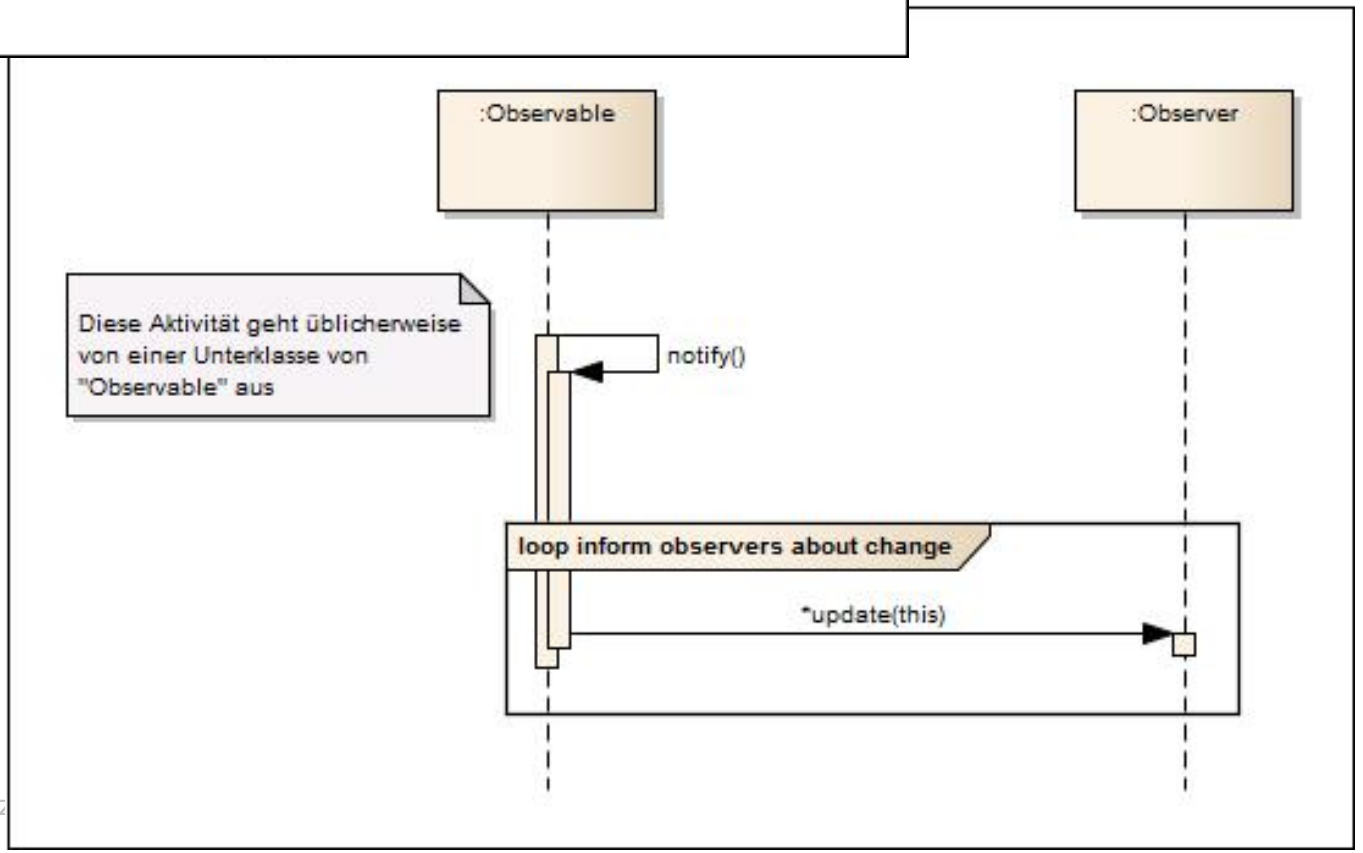
Das Observer-Pattern (auch: Publish/Subscribe, Listener): Observer beim Observable eintragen





Verallgemeinerte Form des Observer-Patterns: eingetragene Observer benachrichtigen





(Der erste) Entwurfsmuster – Katalog

- Erzeugungsmuster
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - *Singleton*
- Strukturmuster
  - Adapter
  - Bridge
  - Decorator (Wrapper)
  - Facade
  - Flyweight
  - Composite
  - Proxy

- Verhaltensmuster
  - Command
  - *Observer*
  - Visitor
  - Interpreter
  - Iterator
  - Memento
  - Template Method
  - Strategy
  - Mediator
  - State
  - Chain of Responsibility

Beschrieben in:
Erich Gamma:
Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software,
Addison-Wesley, 2008

Pattern – Beschreibung nach einheitlichem Schema

Name, alternative Namen	Liefert eine bedeutungsvolle Bezeichnung für das Muster
Zweck	Beantwortet: • Was macht das Muster? • Was ist sein Grundprinzip, sein Zweck? • Welche Probleme löst es? • In welchen Situationen kann es angewandt werden?
Lösung	Beschreibt die Entwurfslösung (Struktur, Interaktion), ihre Beziehungen und Verantwortlichkeiten
Kontext	Beschreibt die Rahmenbedingungen, die erfüllt sein müssen, damit das Muster angewendet werden kann
Konsequenzen	Nennt positive und (sofern bekannt) negative Auswirkungen des Muster-Einsatzes

DHBW Duale Hochschule Baden-Württemberg Mannheim

Pattern-Beschreibung am Beispiel: Observer

Name	Observer
Zweck	Das Observer-Muster ermöglicht einem oder mehreren Objekten automatisch auf die Zustandsänderung eines bestimmten Objektes zu reagieren.
Lösung	Via einer abstrakten Klasse und einem Interface erhalten „interessierte“ Objekte (*Observer*) die Möglichkeit, sich bei dem beobachteten Objekt (*Observable*) zu registrieren. Das beobachtete Objekt informiert die registrierten Objekte über Zustandsänderungen mittels einer *update*-Methode.
Kontext	Zur Entwicklungszeit unbekannte interessierte Objekte können sich ohne enge Kopplung an Zustandsänderungen eines beobachteten Objekts anpassen.
Konsequenzen	+ aktuelle Daten ohne enge Kopplung; beliebige Anzahl Beobachter; einfacher „Testbeobachter“ möglich - keine zentrale Kontrollinstanz: Benachrichtigungskaskaden / Mehrfachbenachrichtigungen möglich - u.U. komplexe Abläufe -> schwer zu debuggen

Beschreibung des Singleton-Pattern

Name	Singleton
Zweck	Stellt sicher, dass genau eine Instanz einer Klasse erzeugt wird.
Lösung	Zugriff auf ein Objekt über eine statische Methode „getInstance“: eine Instanz wird nur erzeugt, falls noch keine existiert.
Kontext	Es ist sichergestellt, dass nur genau eine Instanz einer Klasse erzeugt wird.
Konsequenzen	+ einfach - bei verteilten Anwendungen ist das Erzeugen u.U. problematisch - Löschen ist problematisch, da unter Umständen Clients noch Referenzen halten

Singleton – Beispiel-Implementierung, UML-Darstellung

```
public final class PrinterSpooler {
   private static PrinterSpooler instance = null;
   private PrinterSpooler() {
      ...
   };

   public static PrinterSpooler getInstance() {
      if (instance == null)
         instance = new PrinterSpooler();
      return instance;
   }
}
```

F R A G E N