Beobachter (Observer)

Weiterführende Literatur:

- *Wikipedia-Artikel „Beobachter (Entwurfsmuster)“*
- `https://www.philipphauer.de/study/se/design-pattern/observer.php`
- Gamma u. a., *Design Patterns CD*, Seite 249-257
- Schatten, *Best Practice Software-Engineering*, Kapitel 8.5.1, Seite 269-274
- Eilebrecht und Starke, *Patterns kompakt*, Kapitel 4.7, Seite 70-75
- Siebler, *Design Patterns mit Java*, Seite 269

Zweck

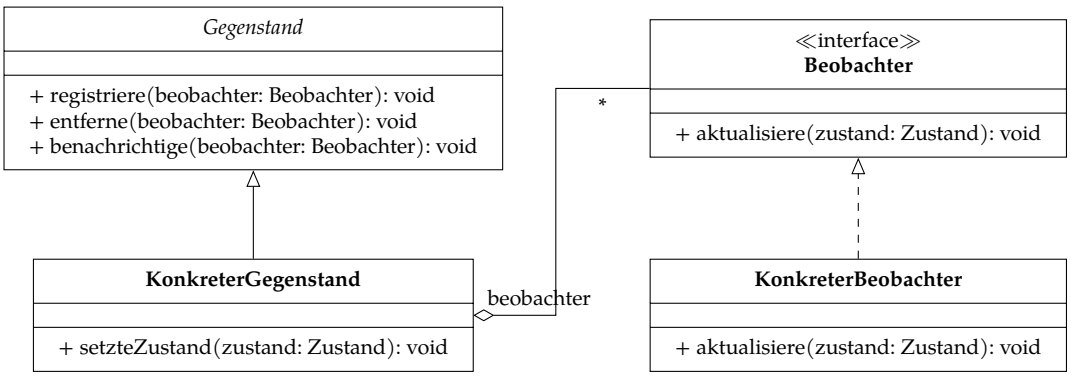
Das Observer-Muster ermöglicht einem oder mehreren Objekten, automatisch auf die *Zustandsänderung* eines bestimmten Objekts *zu reagieren*, um den eigenen Zustand anzupassen.[1]

Zustandsänderung
zu reagieren

Szenario

Zusätzlich zur historischen Darstellung von Verkaufszahlen soll Ihre Software um eine Prognose-Ansicht erweitert werden. Diese soll zum einen jede Viertelstunde die aktuellen Zahlen einbeziehen und zum anderen eine durch den Benutzer gewählte Vorhersagestrategie. Im Gegensatz zur Anzeige des historischen Verlaufs kann sich folglich die Prognose kurzfristig ändern. Genauer gesagt, sobald neue Verkaufszahlen vorliegen oder der Anwender die Strategie ändert. Sie möchten erreichen, dass sich die Anzeige der Prognose automatisch anpasst, sobald sich im Hintergrund Änderungen ergeben.[2]

UML-Diagramm



[1] Eilebrecht und Starke, *Patterns kompakt*, Seite 70.
[2] Eilebrecht und Starke, *Patterns kompakt*, Seite 70.

Akteure

Gegenstand / Subjekt (Subject / Observable) Ein Subjekt (beobachtbares Objekt, auf Englisch publisher, also „Veröffentlicher“, genannt) hat eine Liste von Beobachtern, ohne deren konkrete Typen zu kennen. Es bietet eine Schnittstelle zur An- und Abmeldung von Beobachtern und eine Schnittstelle zur Benachrichtigung von Beobachtern über Änderungen an.[3]

Beobachter (Observer) Die Beobachter (auf Englisch auch subscriber, also „Abonnent“, genannt) definieren eine Aktualisierungsschnittstelle.

konkreter/s Gegenstand / Subjekt (ConcreteSubject / ConcreteObservable) Ein konkretes Subjekt (konkretes, beobachtbares Objekt) speichert den relevanten Zustand und benachrichtigt alle Beobachter bei Zustandsänderungen über deren Aktualisierungsschnittstelle. Es verfügt über eine Schnittstelle zur Erfragung des aktuellen Zustands.

Konkrete Beobachter (ConcreteObserver) Konkrete Beobachter verwalten die Referenz auf ein konkretes Subjekt, dessen Zustand sie beobachten und speichern und dessen Zustand konsistent ist. Sie implementieren eine Aktualisierungsschnittstelle unter Verwendung der Abfrageschnittstelle des konkreten Subjekts.[4]

Allgemeines Code-Beispiel

```java
import java.util.ArrayList;
import java.util.List;

public abstract class Gegenstand {
  private final List<Beobachter> beobachterListe = new ArrayList<Beobachter>();

  public void registriere(Beobachter beobachter) {
    beobachterListe.add(beobachter);
  }

  public void meldeAb(Beobachter beobachter) {
    beobachterListe.remove(beobachter);
  }

  protected void benachrichtige(int zustand) {
    for (Beobachter beobachter : beobachterListe) {
      beobachter.aktualisiere(zustand);
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/entwurfsmuster/beobachter/allgemein/Gegenstand.java`

```java
public class KonkreterGegenstand extends Gegenstand {
  private int zustand;

  public void setzteZustand(int zustand) {
    this.zustand = zustand;
```

[3] Gamma u. a., *Design Patterns CD*, Seite 251.

[4] *Wikipedia-Artikel „Beobachter (Entwurfsmuster)“.*

```
8       // Wenn das Gegenstand die Aktualisierung selbst durchführen soll,
9       // alternativ kann die Methode auch vom Client aufgerufen werden.
10      benachrichtige(zustand);
11    }
12
13    public int gibZustand() {
14      return zustand;
15    }
16  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/entwurfsmuster/beobachter/allgemein/KonkreterGegenstand.java`

```
3   public interface Beobachter {
4     public void aktualisiere(int zustand);
5   }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/entwurfsmuster/beobachter/allgemein/Beobachter.java`

```
3   public class KonkreterBeobachterA implements Beobachter {
4     public void aktualisiere(int zustand) {
5       System.out.println("Konkreter Beobachter A wurde aktualisiert mit " +
        ↪  zustand);
6       // ggf. Modifikationen mit setState().
7     }
8   }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/entwurfsmuster/beobachter/allgemein/KonkreterBeobachterA.java`

```
3   public class KonkreterBeobachterB implements Beobachter {
4     public void aktualisiere(int zustand) {
5       System.out.println("Konkreter Beobachter B wurde aktualisiert mit " +
        ↪  zustand);
6       // ggf. Modifikationen mit setState().
7     }
8   }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/entwurfsmuster/beobachter/allgemein/KonkreterBeobachterB.java`

```
3   public class Klient {
4     public static void main(String[] args) {
5       KonkreterGegenstand konkreterGegenstand = new KonkreterGegenstand();
6       konkreterGegenstand.registriere(new KonkreterBeobachterA());
7       konkreterGegenstand.registriere(new KonkreterBeobachterB());
8       konkreterGegenstand.setzteZustand(77);
9       // Konkreter Beobachter A wurde aktualisiert mit 77
10      // Konkreter Beobachter B wurde aktualisiert mit 77
11    }
12  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/entwurfsmuster/beobachter/allgemein/Klient.java`

Literatur

[1] Karl Eilebrecht und Gernot Starke. *Patterns kompakt. Entwurfsmuster für effektive Softwareentwicklung*. 2019.

[2] Erich Gamma u. a. *Design Patterns CD. Elements of Resuable Object-Oriented Software*. 1995.

[3] Alexander Schatten. *Best Practice Software-Engineering. Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen*. 2010.

[4] Florian Siebler. *Design Patterns mit Java*. 1. Aufl. Hanser, 2014. ISBN: 978-3-446-44111-8.

[5] *Wikipedia-Artikel „Beobachter (Entwurfsmuster)“*. `https://de.wikipedia.org/wiki/Beobachter_(Entwurfsmuster)`.