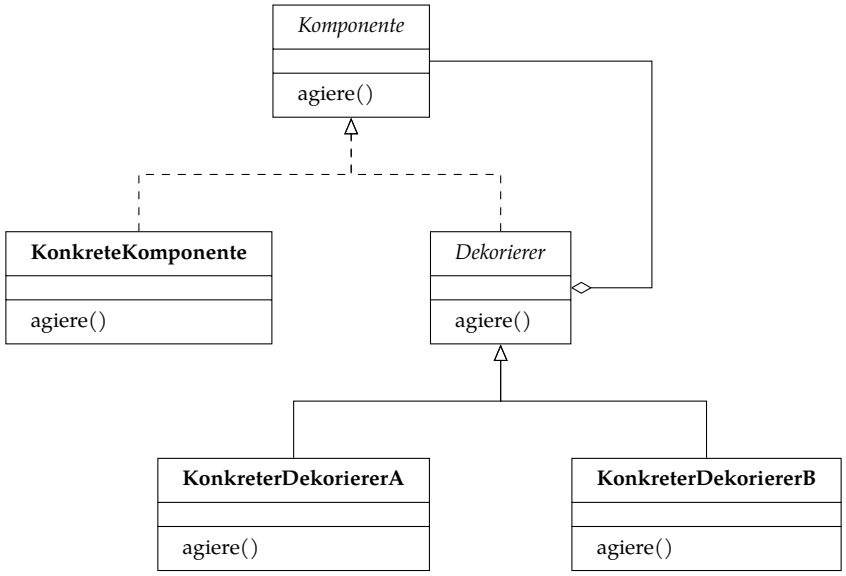
Dekorierer (Decorator)

Weiterführende Literatur:

- *Wikipedia-Artikel „Dekorierer“*
- Gamma u. a., *Design Patterns CD*, Seite 149-156
- `https://www.philipphauer.de/study/se/design-pattern/decorator.php`
- Schatten, *Best Practice Software-Engineering*, Kapitel 8.5.2, Seite 274-278
- Schatten, *Best Practice Software-Engineering*, Seite 274
- Eilebrecht und Starke, *Patterns kompakt*, Kapitel 5.3, Seite 83-86
- Siebler, *Design Patterns mit Java*, Kapitel 22, Seite 269

Zweck

Ein Decorator fügt einer Komponente dynamisch neue Funktionalität hinzu, ohne die Komponente selbst zu ändern.[1]



Szenario

Stellen Sie sich vor, Sie betreiben eine Espressobar. Dort bieten Sie typischerweise neben reinem Espresso auch diverse Kaffeevarianten an: echten Kaffee oder ohne Koffein, mit heißer (Latte Macciato) oder geschäumter Milch (Cappuccino), mit Sahne, mit einer Kugel Eis, mit Likör oder doppelt (Doppio). Wenn Sie diese Kombinationen alle explizit modellieren, erhalten Sie eine unübersichtlich große Zahl von Klassen. Erweiterungen des Angebotes (etwa mit Ahorn- oder Nussaroma) lassen die Anzahl Klassen drastisch weiter wachsen.[3]

[1] Eilebrecht und Starke, *Patterns kompakt*, Seite 83.
[3] Eilebrecht und Starke, *Patterns kompakt*, Seite 83.

Allgemeines Code-Beispiel

```java
/**
 * Abstrakte Klasse oder Interface je nach Bedarf.
 */
public abstract class Komponente {

  public abstract void agiere();
}
```

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

```java
public class KonkreteKomponente extends Komponente {
  public void agiere() {
    System.out.println("KonkreteKomponente agiert.");
  }
}
```

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

```java
public abstract class Dekorierer extends Komponente {
  protected Komponente komponente;

  /**
   * Konstruktor zum komfortablen Initiieren am Client
   *
   * @param komponente Die Komponente.
   */
  public Dekorierer(Komponente komponente) {
    this.komponente = komponente;
  }
  // agiere() wird nicht implementiert.
}
```

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

```java
class KonkreterDekoriererA extends Dekorierer {
  public KonkreterDekoriererA(Komponente komponent) {
    super(komponent);
  }

  /**
   * agiere() der dekorierten Kompontente aufrufen vor
   * der nach der eigenen Funktionalität.
   */
  public void agiere() {
    komponente.agiere();
    // eigene Funktionalität:
    System.out.println("KonkreterDekoriererA agiert!");
  }

}
```

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

```java
/**
 * KonkreterDekoriererB fügt der Komponente einen neuen Zustand hinzu.
 */
class KonkreterDekoriererB extends Dekorierer {

  private int neuerZustand;

  public KonkreterDekoriererB(Komponente komponente) {
```

```
11      super(komponente);
12      neuerZustand = 999;
13    }
14
15    public void agiere() {
16      komponente.agiere();
17      System.out.println("KonkreterDekoriererB agiert mit dem neuen Zustand: " +
        ↪  neuerZustand);
18    }
19  }
```

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

```
3   public class Klient {
4     public static void main(String[] args) {
5       Komponente komponente = new KonkreteKomponente();
6       komponente.agiere();
7       // KonkreteKomponente agiert.
8
9       komponente = new KonkreterDekoriererA(komponente);
10      komponente.agiere();
11      // KonkreteKomponente agiert.
12      // KonkreterDekoriererA agiert!
13
14      komponente = new KonkreterDekoriererB(new KonkreteKomponente());
15      komponente.agiere();
16      // KonkreteKomponenteA agiert.
17      // KonkreterDekoriererB agiert mit dem neuen Zustand: 999
18    }
19  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/entwurfsmuster/dekorierer/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 „Dekorierer“*. `https://de.wikipedia.org/wiki/Dekorierer`.