RUHR-UNIVERSITÄT BOCHUM
Lehrstuhl für Informatik im Bauwesen

RUB

Objektorientierte Modellierung

Sommersemester 2023

Command-Muster

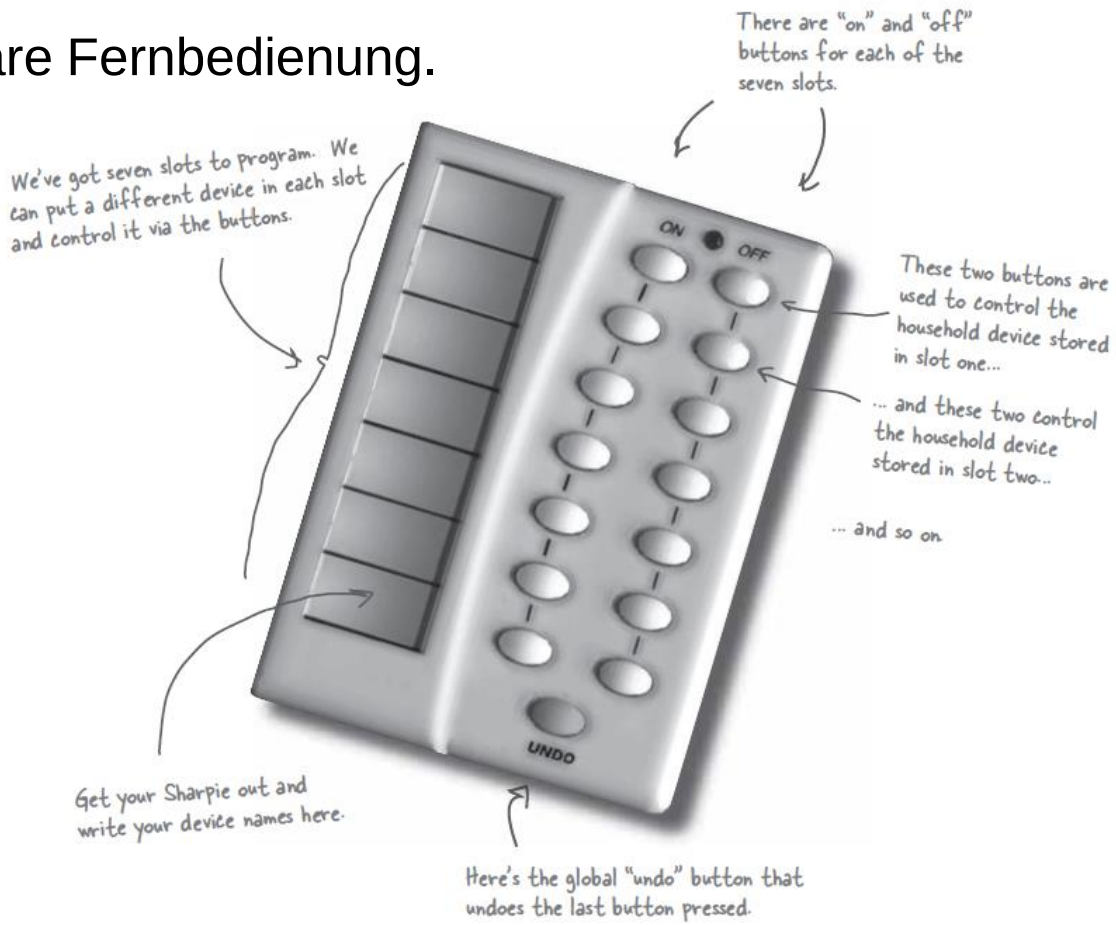
Command-Muster

Aufrufe kapseln

Gegeben ist eine programmierbare Fernbedienung.

Es gibt eine bestimmte Anzahl von programmierbaren Plätzen für externe Geräte.

Für jeden Platz gibt es einen AN- und einen AUS-Knopf zur Steuerung.



Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

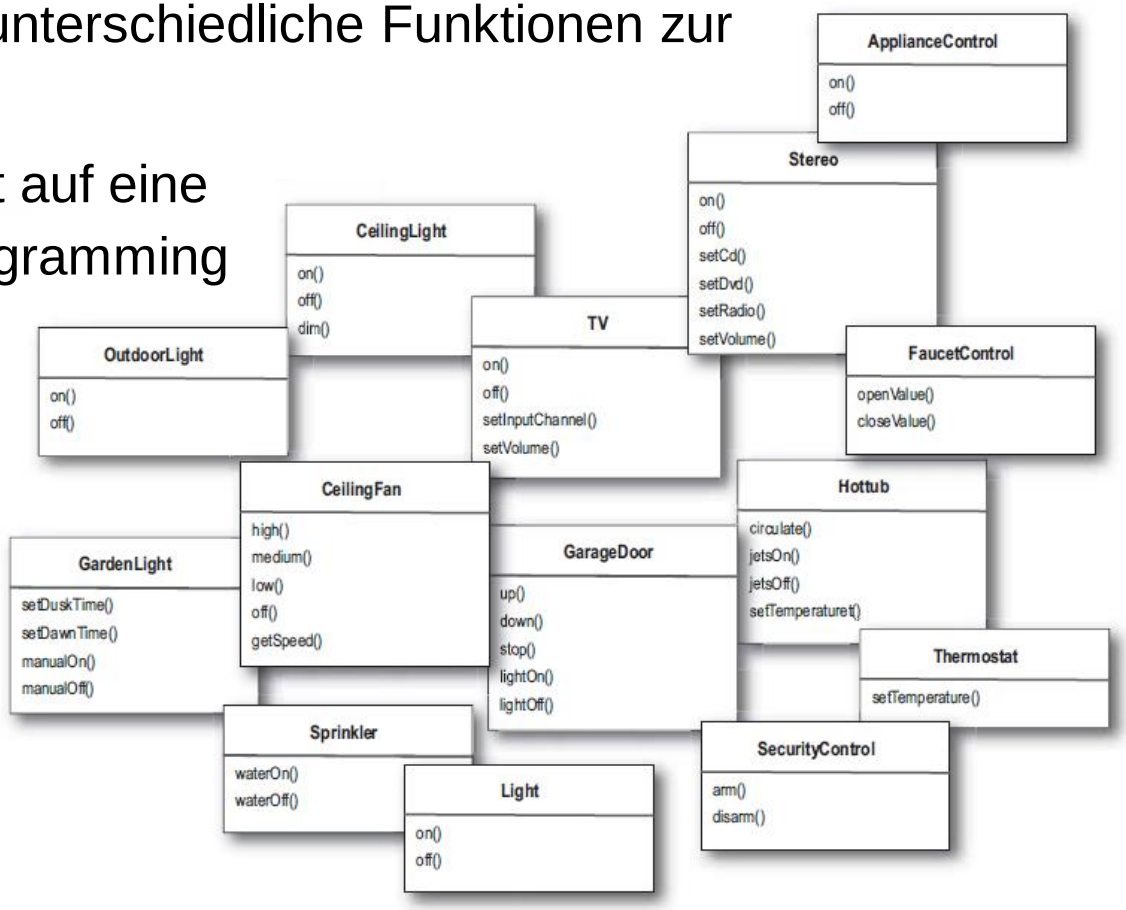
Command-Muster

Aufrufe kapseln

Die steuerbaren Geräte stellen unterschiedliche Funktionen zur Steuerung zur Verfügung.

Die Hersteller konnten sich nicht auf eine einheitliche API (Application programming interface) einigen.

Eine Änderung der gegebenen Klassen ist nicht möglich.



Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Command-Muster

Verhaltensmodellierung

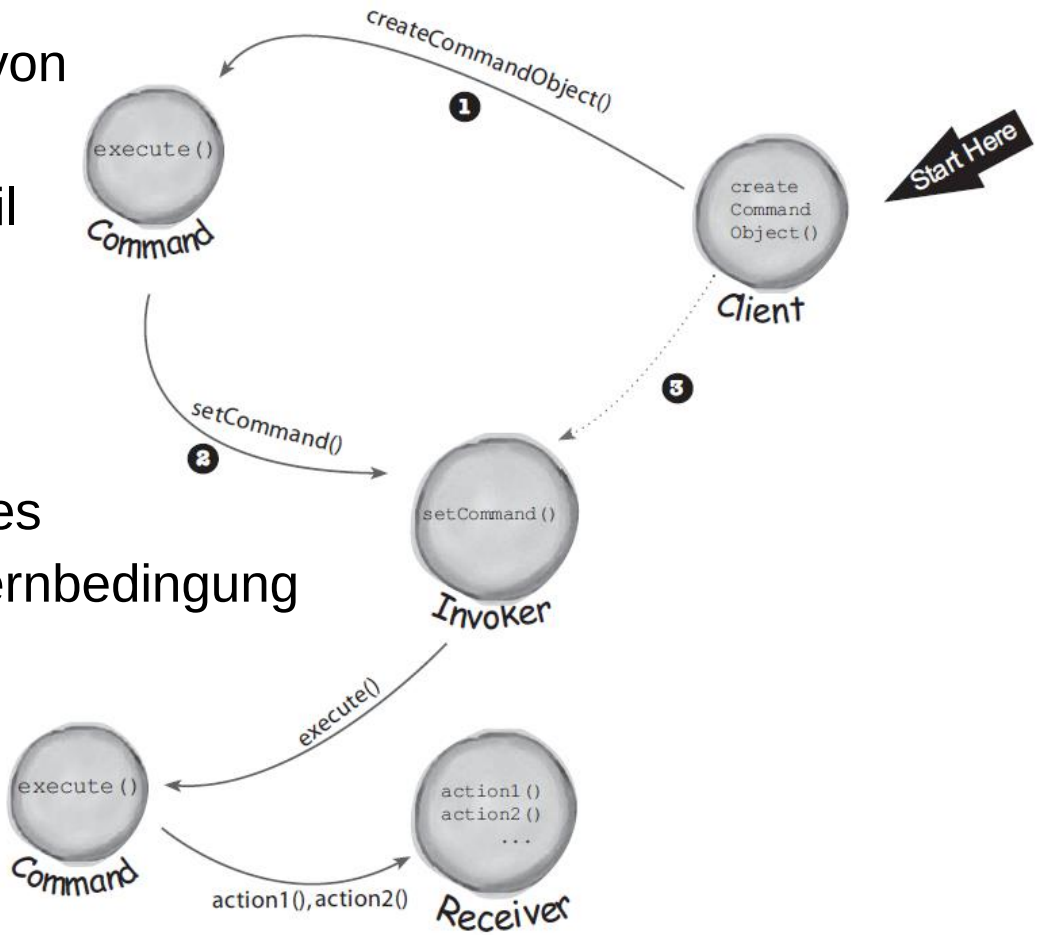
Neben der statischen Modellierung von Klassen sollte auch das dynamische Verhalten einer Anwendung im Detail beschrieben werden.

Sequenz A

Ein Anwender (Client) erzeugt einen Befehl (Command) für ein bestimmtes Gerät und speichert dieses in der Fernbedingung (Invoker).

Sequenz B

Ein Anwender drückt einen Knopf und führt somit den Befehl auf dem Gerät (Receiver) aus.



Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

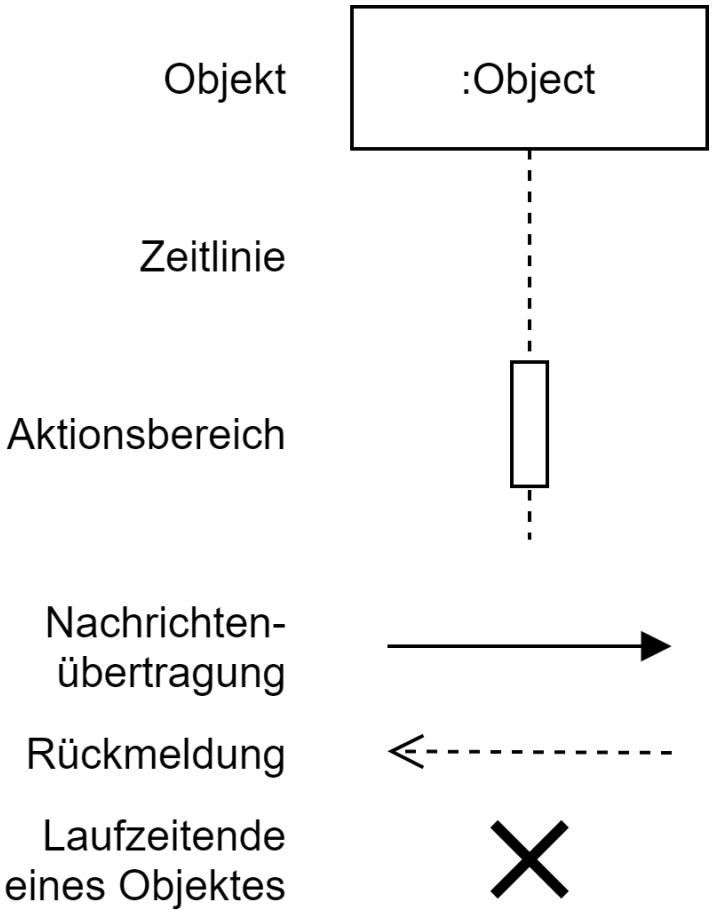
Command-Muster

Verhaltensmodellierung

Verhalten bzw. die Kommunikation von Objekten kann mit Hilfe eines Sequenzdiagramms modelliert werden:

Das Sequenzdiagramm beschreibt die zeitliche Abfolge von Interaktionen zwischen einer Menge von Objekten innerhalb eines zeitlich begrenzten Kontextes.

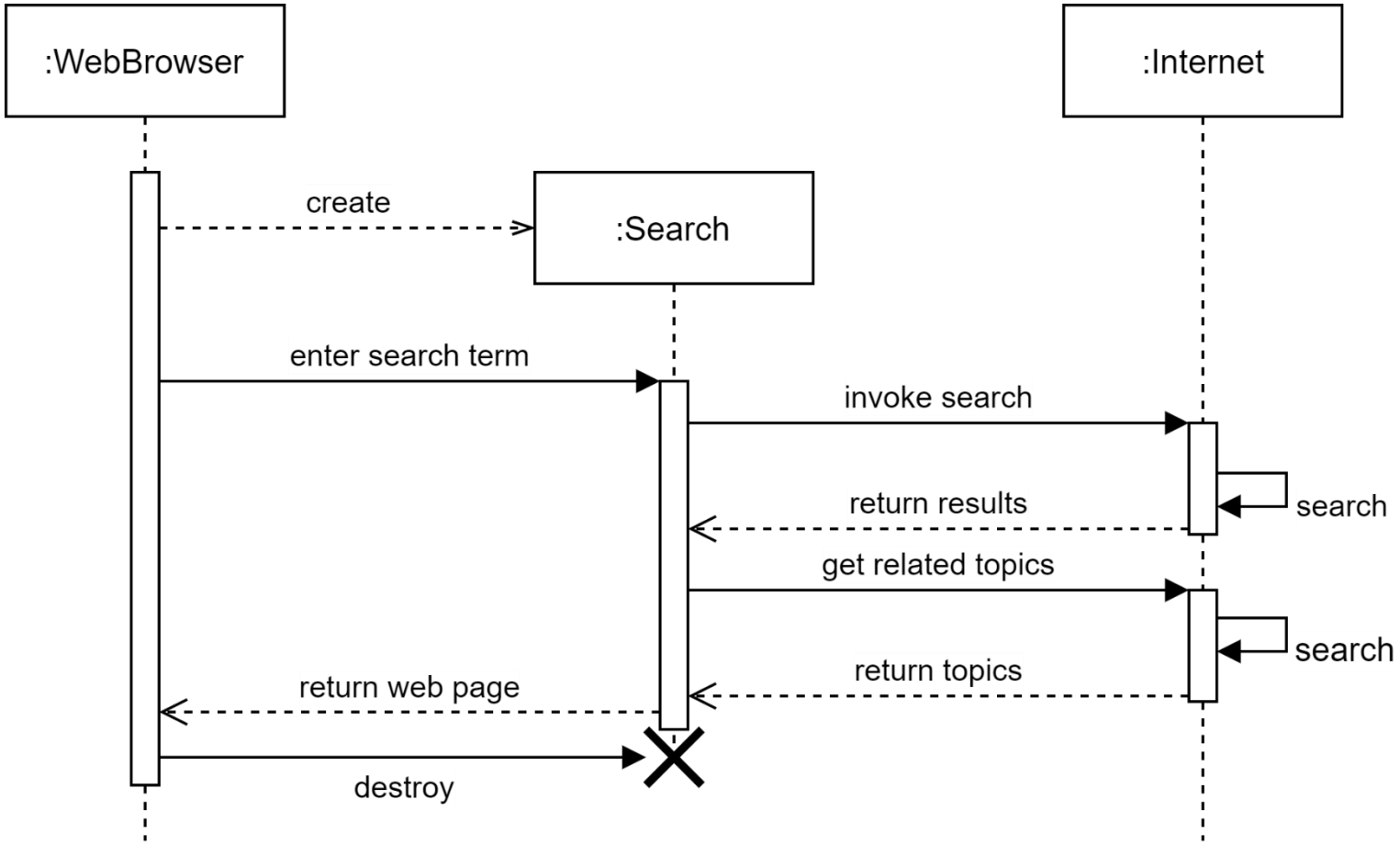
Folgende Elemente kann ein Sequenzdiagramm besitzen:



Command-Muster

Sequenzdiagramm

Beispiel:



Command-Muster

Sequenzdiagramm

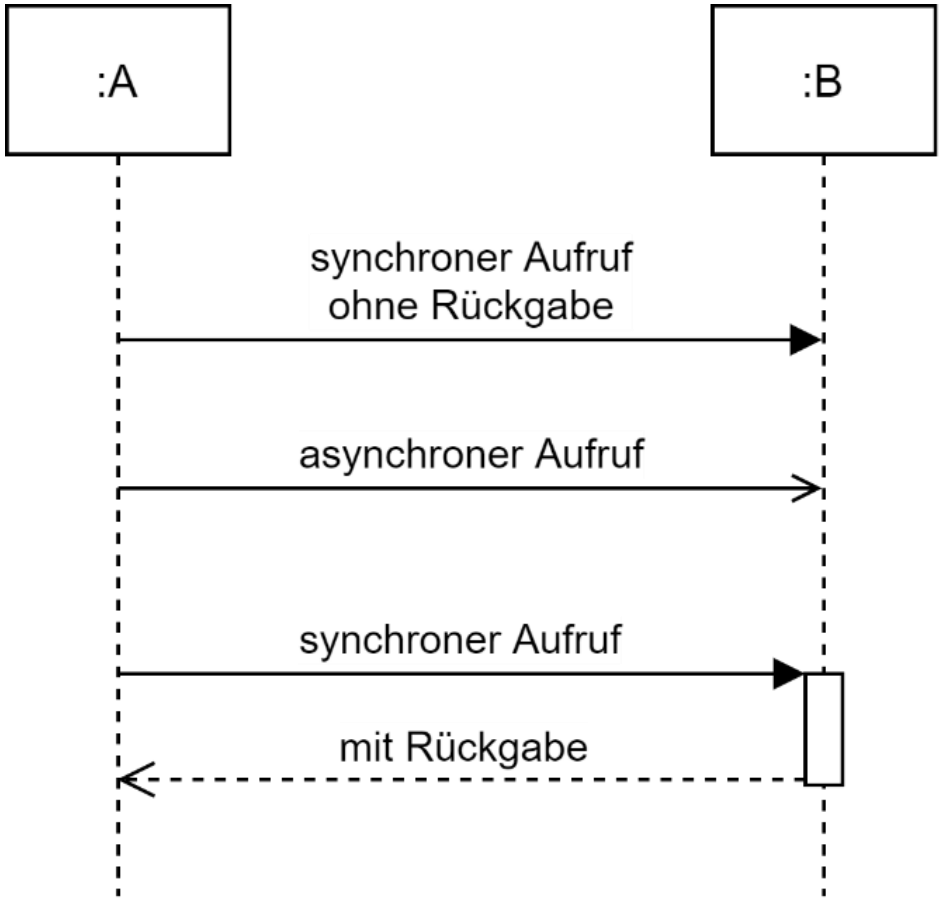
- Objekte (als Rechteck dargestellt) haben eine Lebenslinie (gestrichelte Linie), von der aus sie mit anderen Objekten kommunizieren können (Zeit verläuft von oben nach unten)
  - Sequenzdiagramme zeigen die Laufzeit von Objekten an
  - Bereiche in dem ein Objekt aktiv ist werden durch Balken angezeigt
- Nachrichten zwischen Elementen sind z.B. Funktionsaufrufe oder Netzwerkanfragen
  - Verdeutlicht Aufrufhierarchie und zeitliche Abfolge von Aufrufen

Command-Muster

Sequenzdiagramm

Bei einem synchronen Aufruf wartet der Absender bis der Empfänger fertig ist. Eine Rückgabe ist optional.

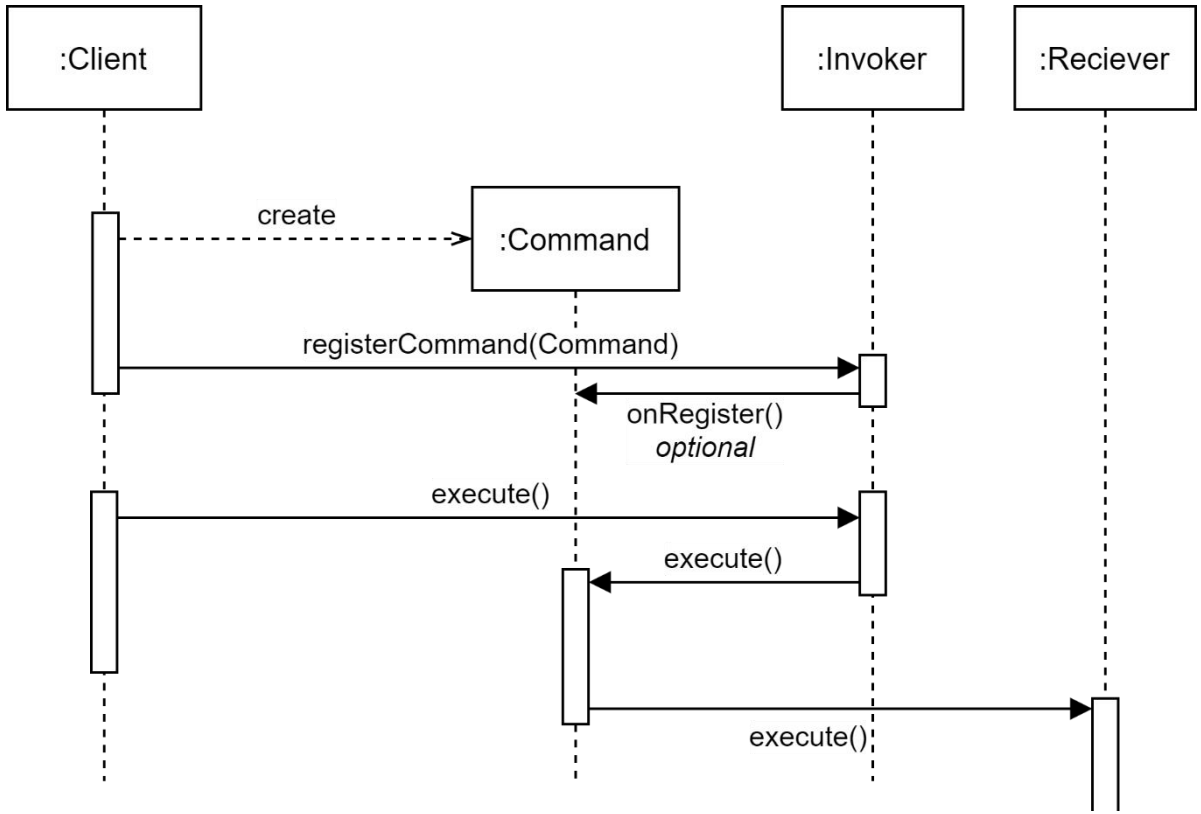
Bei einem asynchronen Aufruf arbeiten Sender und Empfänger parallel weiter.



Command-Muster

Sequenzdiagramm

Beispiel Fernbedienung:



Command-Muster

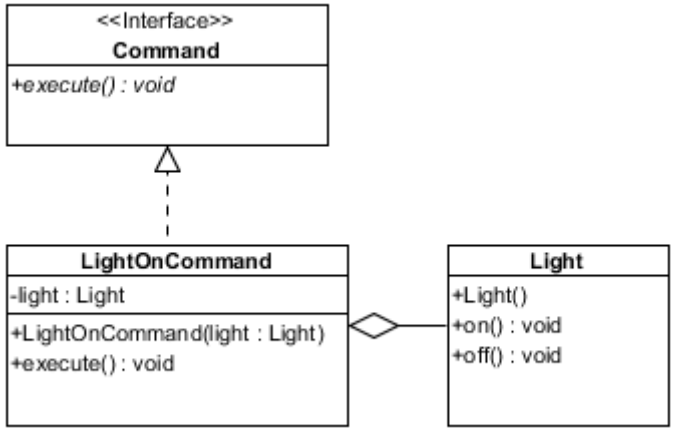
Command-Schnittstelle

Alle einzelnen Befehl-Objekte sollen die gleiche Schnittstelle implementieren. Die Schnittstelle besitzt nur eine Methode für die Ausführung des Befehls.

```
public interface Command {
    public void execute();
}
```

Die einzelnen Befehle der Gerätehersteller werden jetzt in neue Klassen gekapselt.

```
public class LightOnCommand implements Command {
    Light light;

    public LightOnCommand(Light light) {
        this.light = light;
    }

    public void execute() {
        this.light.on();
    }
}
```

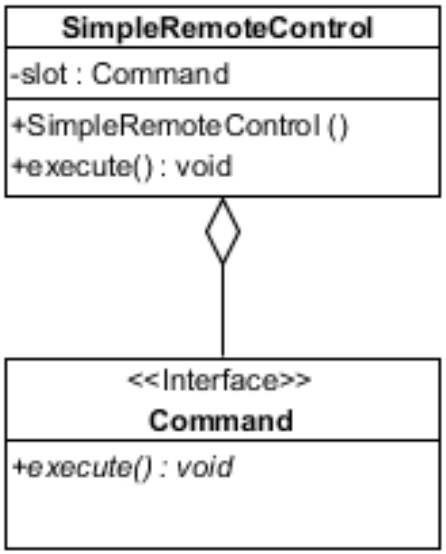


Command-Muster

Einfache Fernbedienung

Es wird angenommen, dass die Fernbedienung aktuell nur mit einem Knopf und einem Speicherplatz ausgestattet ist. Somit kann nur ein Befehl aufgenommen werden

```
public class SimpleRemoteControl {
    Command slot;

    public SimpleRemoteControl() {}

    public void setCommand(Command command) {
        this.slot = command;
    }

    public void execute() {
        this.slot.execute();
    }
}
```



Command-Muster

Einfache Fernbedienung

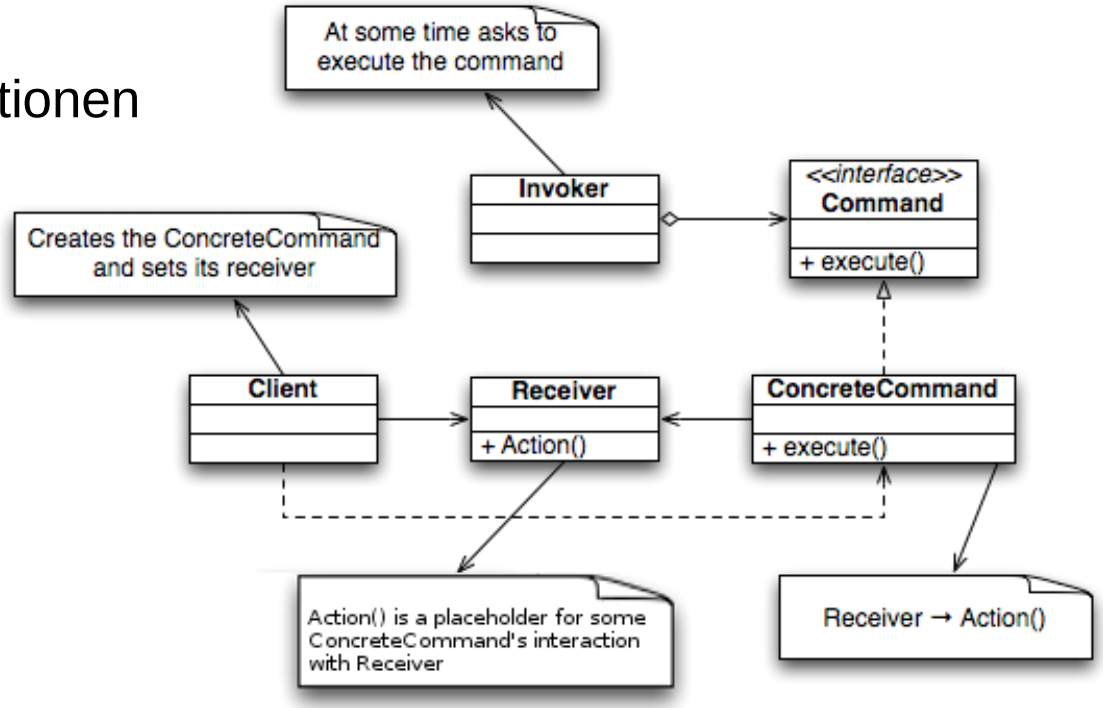
Testprogramm

```
public class RemoteControlTest {
    public static void main(String[] args) {

        SimpleRemoteControl remote = new SimpleRemoteControl();
        Light light = new Light();

        LightOnCommand lightOn = new LightOnCommand(light);

        remote.setCommand(lightOn);
        remote.execute();
    }
}
```

Command-Muster

Das Command-Muster kapselt einen Auftrag als ein Objekt und ermöglicht es so, andere Objekte mit verschiedenen Aufträgen zu parametrisieren, Aufträge in Warteschlangen einzureihen oder zu protokollieren.

Dadurch kann auch das Rückgängigmachen von Operationen unterstützt werden.



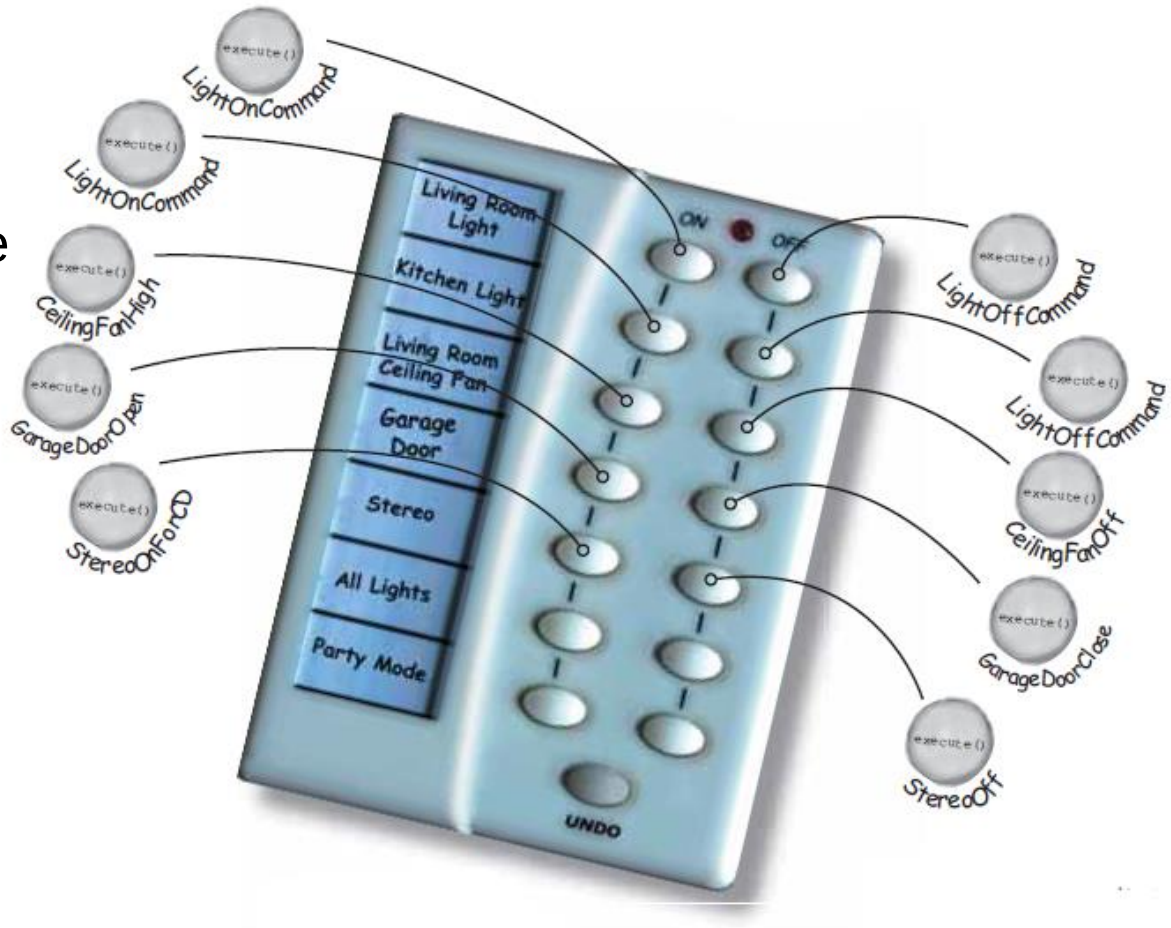
Command-Muster

Multi-Fernbedienung

Im nächsten Schritt wird eine Fernbedienung mit nx2 Speicherplätzen vorgesehen.

Es sollen verschiedene Geräte damit gesteuert werden können (z.B. Licht, Musik, etc.)

RemoteControl
-commands : Command[][]
+RemoteControl(n : int) +setCommand(slot : int, on : Command, off : Command) +on(slot : int) +off(slot : int)



Command-Muster

Multi-Fernbedienung

Implementierung (Auszug)

```
public class RemoteControl {
    Command[][] commands;

    public RemoteControl(int n) {
        this.commands = new Command[n][2];

        Command noCommand = new NoCommand();
        for (int i = 0; i < n; i++) {
            this.commands[i][0] = noCommand;
            this.commands[i][1] = noCommand;
        }
    }
...
}
```

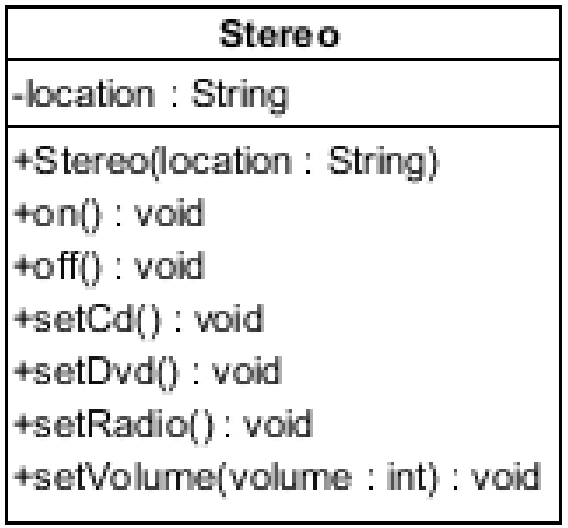
```
public class NoCommand implements Command {

    public void execute() { }
}
```

Command-Muster

Komplexe Befehle

Es können natürlich auch mehrere Elemente in einem Befehl zusammengefasst werden. In diesem Beispiel soll die Stereoanlage geschaltet und CD-Player aktiviert werden.

```
public class StereoOnWithCDCommand implements Command {
    Stereo stereo;

    public StereoOnWithCDCommand(Stereo stereo) {
        this.stereo = stereo;
    }

    public void execute() {
        this.stereo.on();
        this.stereo.setCD();
    }
}
```



Command-Muster

Befehle rückgängig machen

Eine Rückgängigmachen-Funktionalität gehört zum entsprechenden Befehl, da nur diese weiß, welche Aktion beim Gerät ausgeführt werden muss. Somit muss das entsprechende Interface erweitert werden.

```
public interface Command {
    public void execute();
    public void undo();
}
```

Entsprechend müssen jetzt die einzelnen Klassen auch die neue Methode implementieren

```
public class LightOnCommand implements Command {
...

    public void undo() {
        this.light.off();
    }
}
```

Command-Muster

Multi-Fernbedienung

Implementierung (Auszug)

```
public class RemoteControl {
    Command[][] commands;
    Command undoCommand;

    public RemoteControl(int n) {
        this.commands = new Command[n][2];

        Command noCommand = new NoCommand();
        for (int i = 0; i < n; i++) {
            this.commands[i][0] = noCommand;
            this.commands[i][1] = noCommand;
        }
        this.undoCommand = noCommand;
    }

    public void undo() {
        this.undoCommand.undo();
    }
...
}
```

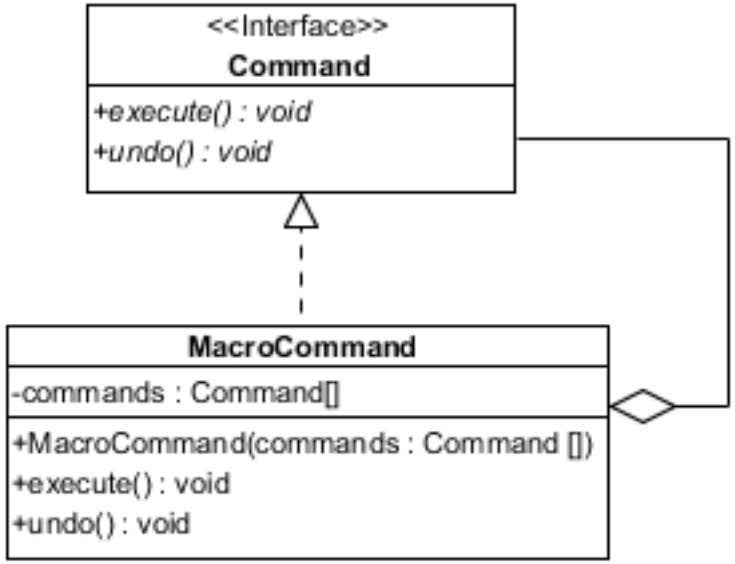
```
...
public void on(int slot) {
    this.commands[slot][0].execute();
    this.undoCommand = this.commands[slot][0];
}

public void off(int slot) {
    this.commands[slot][1].execute();
    this.undoCommand = this.commands[slot][1];
}
...
```

Command-Muster

Der Party-Modus

Im nächsten Schritt soll ein Party-Modus für die Fernbedienung programmiert werden. Dabei sollen verschiedene Geräte mit einem Knopf in bestimmte Zustände versetz werden z.B. Licht an, Stereoanlage an, Lautstärke auf 10 setzen und Ventilator an.

Damit diese Einstellungen individuell vorgenommen werden können, wird ein sogenannter Makro-Befehl umgesetzt, der eine Liste von einzelnen Befehlen ausführt.



Command-Muster

Einfache Fernbedienung

Testprogramm

```
public class RemoteControlTest {
    public static void main(String[] args) {
        RemoteControl remoteControl = new RemoteControl(1);
        ...
        Command[] partyOn = { lightOn, stereoOn, stereoVolumeMax, ceilingFanHigh };
        Command[] partyOff = { lightOff, stereoOff, ceilingFanOff };

        MacroCommand partyOnMacro = new MacroCommand(partyOn);
        MacroCommand partyOffMacro = new MacroCommand(partyOff);

        remoteControl.setCommand(0, partyOnMacro, partyOffMacro);

        System.out.println(remoteControl);
        System.out.println("--- Pushing Macro On---");
        remoteControl.on(0);
        System.out.println("--- Pushing Macro Off---");
        remoteControl.off(0);
    }
}
```

Command-Muster

Fazit

Das Command-Muster kapselt einen Auftrag als ein Objekt und ermöglicht Objekte mit verschiedenen Aufträgen zu parametrisieren. Des Weiteren können Aufträge in Warteschlangen eingereiht oder protokolliert werden. Auch das Rückgängigmachen von Operationen wird ermöglicht.

Makro-Befehle sind eine Erweiterung des Command-Musters, die es ermöglichen, mehrere Befehle aufzurufen.

Befehle können auch verwendet werden, um Logger- und Transaktionssystem zu implementieren.

RUHR-UNIVERSITÄT BOCHUM
Lehrstuhl für Informatik im Bauwesen

RUB

Programmierung und Programmiersprachen

Sommersemester 2023

Decorator-Muster

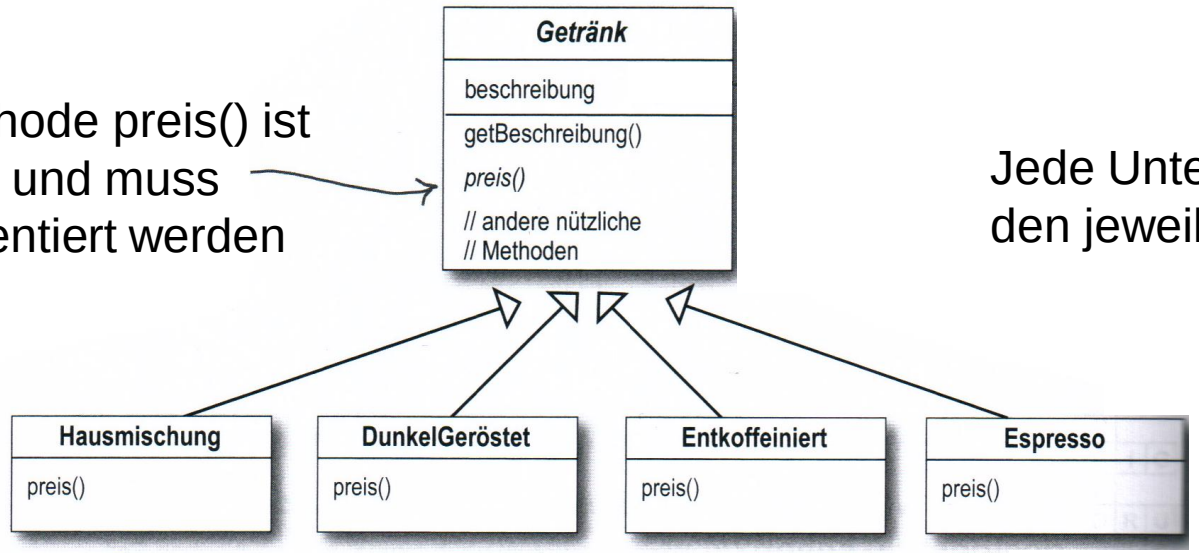
Decorator-Muster

Probleme mit minimalen Spezialisierungen

Gegeben sind folgende Klassen zur Preisermittlung von Getränkebestellungen (hier Kaffee).

Abstrakte Klasse von der alle Getränke abgeleitet werden

Die Methode preis() ist abstrakt und muss implementiert werden



Jede Unterklasse berechnet den jeweiligen Preis

Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

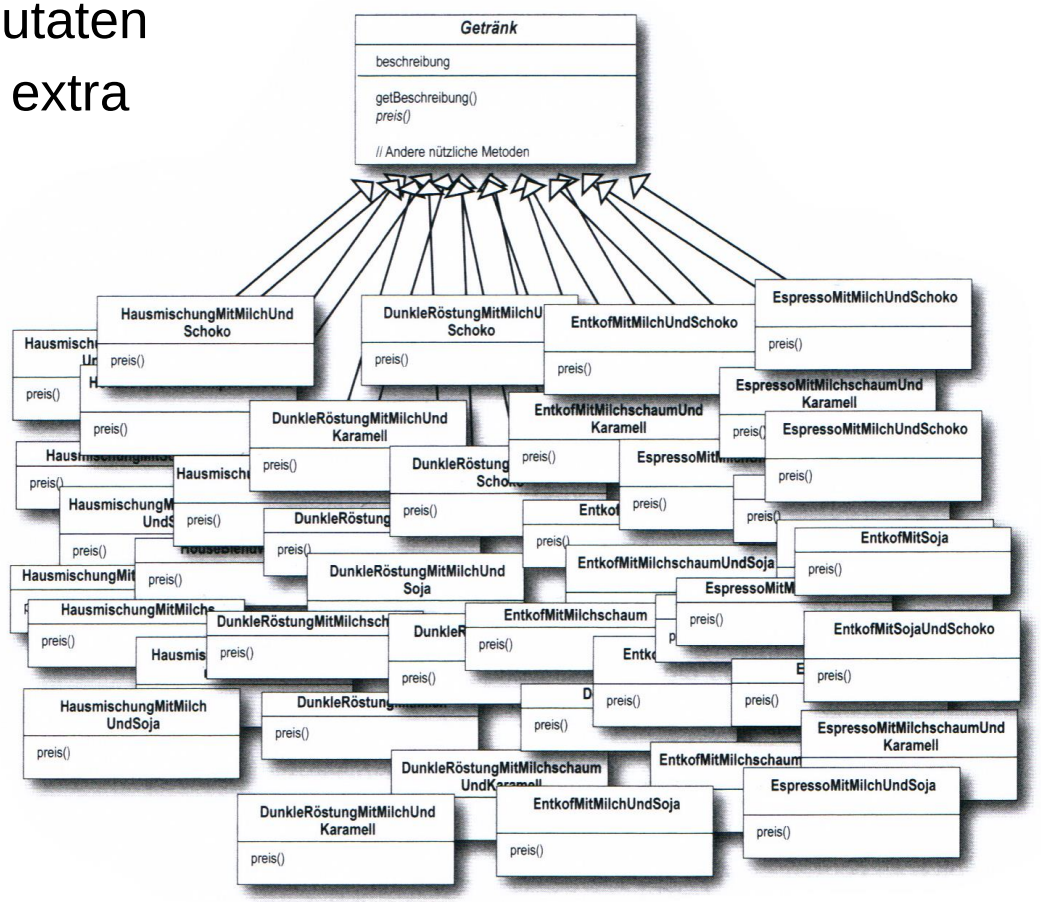
Decorator-Muster

Probleme mit minimalen Spezialisierungen

Zum Kaffee können verschiedene Zutaten bestellt werden. Je Zutat wird dabei extra berechnet.

Zutaten

- *heiße Milch*
- *heiße Sojamilch*
- *Schokostreusel*
- *Karamellsirup*
- *Milchschaum*
- *…*

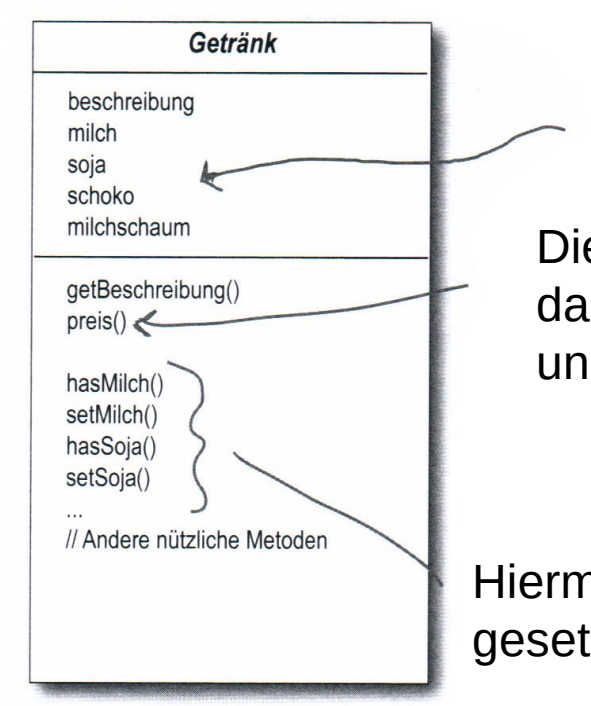


Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Decorator-Muster

Spezialisierung mit Attributen

Die einzelnen Zutaten könnten mit Hilfe von booleschen Variablen in der Basisklasse definiert werden.



Neue boolesche Attribute für die einzelnen Zutaten

Die Methode preis() prüft nun das Vorhandensein der Zutaten und berechnet den Preis

Hiermit können die Zutaten gesetzt und abgefragt werden

Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Decorator-Muster

Implementierung (Auszug)

```
public class Hausmischung extends Getränk {
    ...
    // Preis berechnen
    public double preis() {
        // Standard
        double preis = 0.89;
        // Einzelne Zutaten hinzufügen
        if (super.hasMilch()) {
            preis += 0.10;
        }
        if (super.hasMilchschaum()) {
            preis += 0.10;
        }
        ...
        return preis;
    }
    ...
}
```

Decorator-Muster

Spezialisierung mit Attributen

Welche Probleme könnte diese Umsetzung hervorrufen (siehe Strategie-Muster)?

- Preisänderungen bei den Zutaten erfordert die Anpassung aller Getränke-Klassen
- Neue Zutaten erfordern eine Erweiterung der Basisklasse und Anpassung aller Kind-Klassen
- Nicht alle Zutaten passen zu allen Getränken (z.B. neues Getränk Eistee)
- Ein Kunde bestellt einen Kaffee mit Doppelschoko

Wie können Klassen für Erweiterungen offen, aber für Veränderungen geschlossen sein?

Das Decorator-Muster

Bei der Bearbeitung einer Getränkebestellung wird das Basisgetränk um Zutaten ergänzt bzw. es wird dekoriert.

Vorgehensweise:

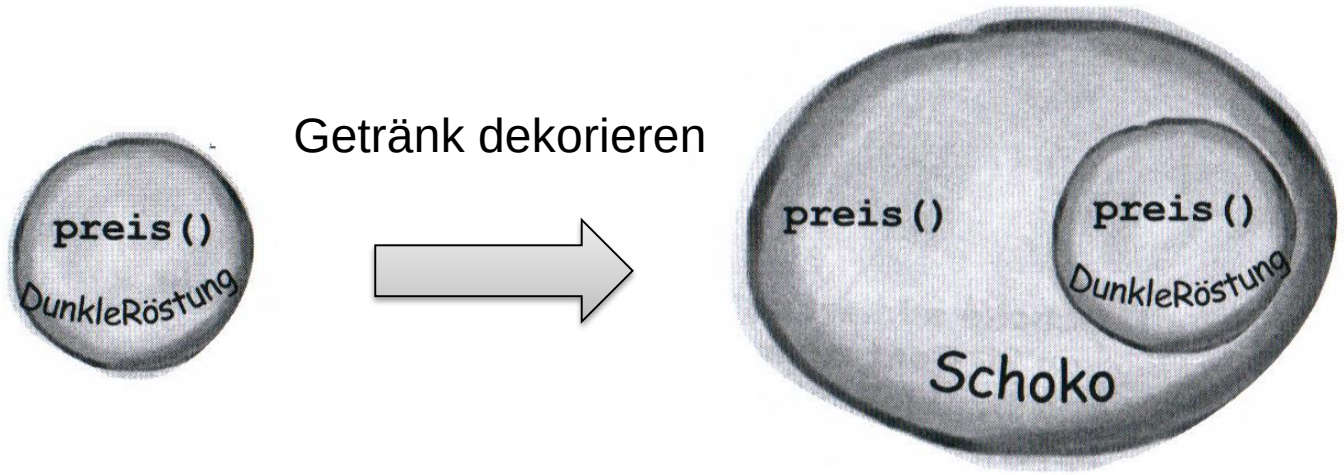
- Erzeuge ein DunkleRöstung-Objekt
- Dekoriere es mit einem Schoko-Objekt
- Dekoriere es mit einem Milchschaum-Objekt
- Berechne den Preis der einzelnen Objekte

Die Dekorierer-Objekte werden häufig als Wrapper bezeichnet.

Decorator-Muster

Getränk mit Dekorierern aufbauen

Das Basisgetränk, hier DunkleRöstung, kennt seinen Preis und wird umschlossen durch ein Schoko-Objekt, welches wiederum ein Getränk ist und eine Preis-Methode besitzt (für die Implementierung wichtig).

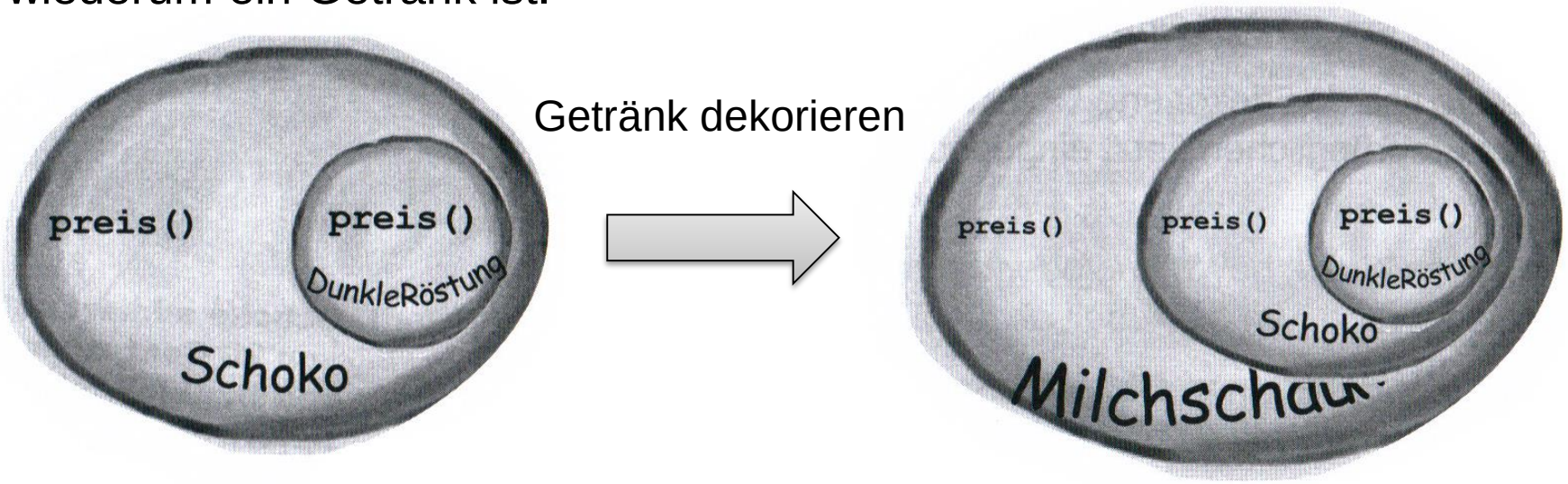


Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Decorator-Muster

Getränk mit Dekorierern aufbauen

Soll weiterhin Milchschaum hinzugefügt werden, wird das zuvor dekorierte Getränk durch ein weiteres Objekt umschlossen, welches wiederum ein Getränk ist.



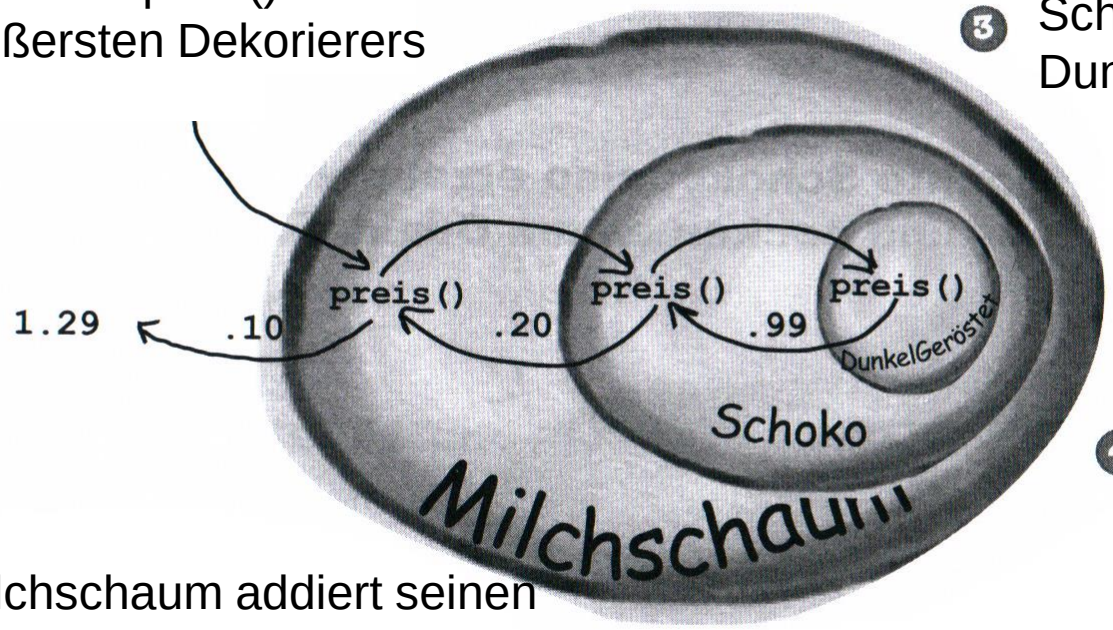
Achtung: Zutaten-Objekte erfordern bei der Erstellung immer ein Getränk (dekoriert oder auch nicht).

Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Decorator-Muster

Berechnung des Gesamtpreises

Der Aufruf der Methode preis() erfolgt über das letzte Objekt.



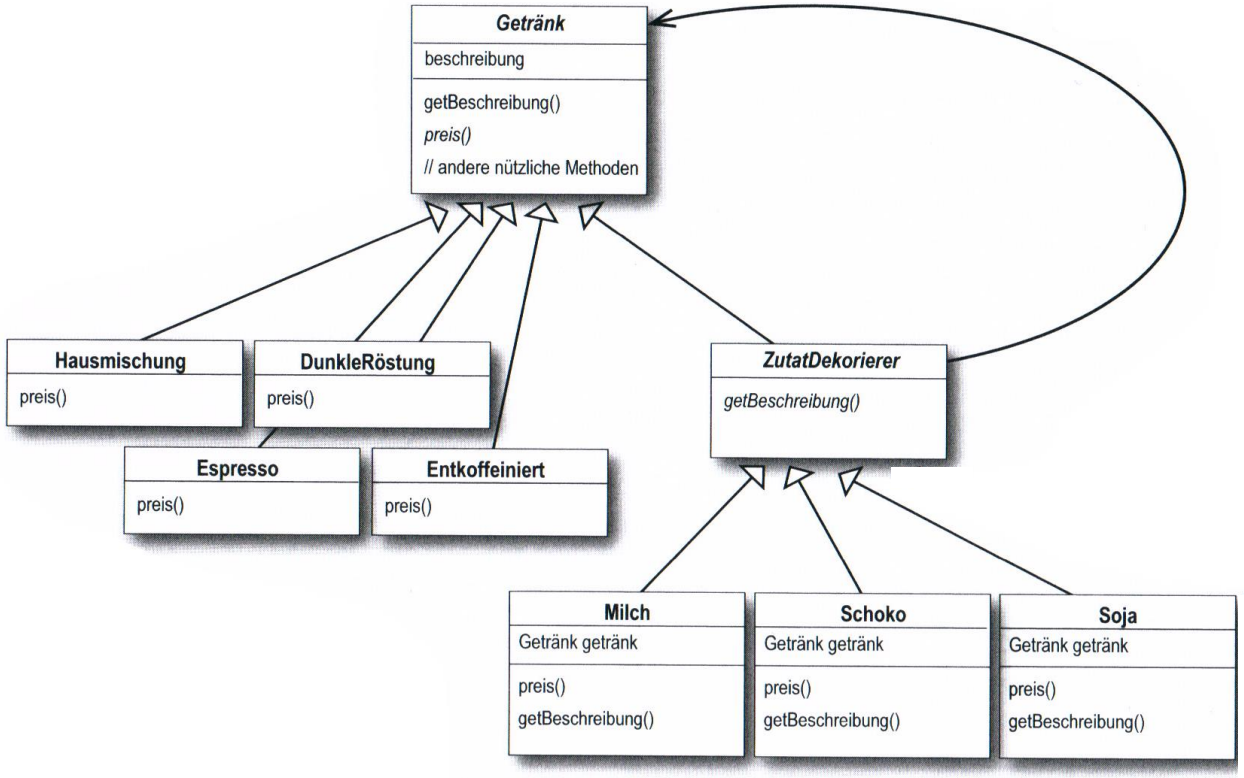
1. Methode preis() des Äußersten Dekorierers
2. Milchschaum ruft preis() auf Schoko auf
3. Schoko ruft preis() auf DunkleRöstung auf
4. DunkleRöstung liefert den Basispreis
5. Schoko addiert seinen Preis
6. Milchschaum addiert seinen Preis

Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Decorator-Muster

Getränke-Framework

Es wird eine abstrakte Methode für den Dekorierer erstellt, welche die Basis-Klasse für Getränke erweitert und ein (bzw. dekoriertes) Getränk enthält.



Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

Decorator-Muster

Implementierung (Auszug)

```
public abstract class ZutatDekorierer extends Getränk {
   protected Getränk getränk;
   public abstract String getBeschreibung();
}

public class Schoko extends ZutatDekorierer {
   // Dem Konstruktur muss ein Getränk zum dekorieren übergeben werden
   public Schoko(Getränk getränk) {
      super.getränk = getränk;
   }

   public String getBeschreibung() {
      return super.getränk.getBeschreibung()+ ", Schoko";

   public double preis() {
      return 0.20 + super.getränke.preis();
   }
}
```

Decorator-Muster

Testprogramm

```
public abstract class KaffeeBestllung {

   public static void main(String[] args) {

      // Hausmischung mit heißer Milch, Doppelkaramell und Milchschaum
      Getränk getränk1 = new Espresso();           //  0.89
      getränk1 = new Milch(getränk1);              // +0.10
      getränk1 = new Karamell(getränk1);           // +0.20
      getränk1 = new Karamell(getränk1);           // +0.20
      getränk1 = new Milchschaum(getränk1);        // +0.10

      // Preis ausgeben
      System.out.println(getränk1.getBeschreibung() + " " +
         getränk1.preis() + " EURO");
   }
}
```

Decorator-Muster

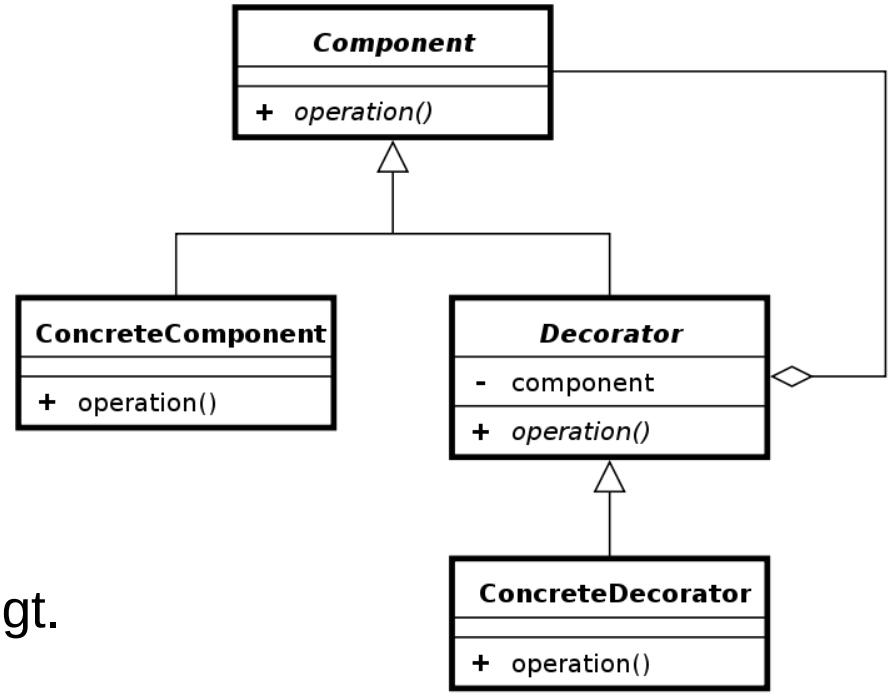
Fazit

Das Decorator-Muster ist eine Alternative zur Spezialisierung von Klassen.

Es werden eigene Klassen zum Dekorieren erzeugt.

Dekorierer-Klassen sind Unterklassen der Klasse, die dekoriert werden soll.

Durch ein Dekorierer werden einem Objekt zusätzliche Funktionen hinzugefügt.



Achtung: Sollte nicht übermäßig verwendet werden, es könnten sehr viele Objekte erzeugt werden. Des Weiteren wird der Code unübersichtlicher.