CAMPUS 02 GRAZ
FACHHOCHSCHULE DER WIRTSCHAFT

SOFTWARE DESIGN FOLIENSATZ 5

Command + Adapter

Bernhard Fuchs
ZAM - WS 2025/26

TERMINE

0004VD2005 SOFTWARE DESIGN (29,75UE IL, WS 2025/26)

Gruppe

	Tag	Datum	von	bis	Ort	Ereignis	Termintyp	Lerneinheit	Vortragende*r	Anmerkung
	Standardgruppe									
	Do	22.01.2026	08:15	11:45	CZ106	Abhaltung	fix	**1 Organisation + Strategy Pattern**		
	Do	29.01.2026	08:15	11:45	CZ106	Abhaltung	fix	**2 Decorator Pattern + Singleton**		
	Do	05.02.2026	12:30	16:00	CZ106	Abhaltung	fix	**3 Observer**		
	Fr	06.02.2026	08:15	12:15	CZ106	Abhaltung	fix	**4 Factory**		
Heute	Do	12.02.2026	08:15	13:00	CZ106	Abhaltung	fix	**5 Command + Adapter**		
	Fr	13.02.2026	08:15	12:15	CZ106	Abhaltung	fix	**6 Facade, Template, Iterator** mit Christian Hofer!		
	Do	19.02.2026	08:15	11:45	CZ106	Abhaltung	fix	**Wiederholung**		
	Fr	20.02.2026	08:15	12:15	CZ106	Prüfungstermin	fix	**Prüfung**		

EIGENRECHERCHE

Command-Pattern

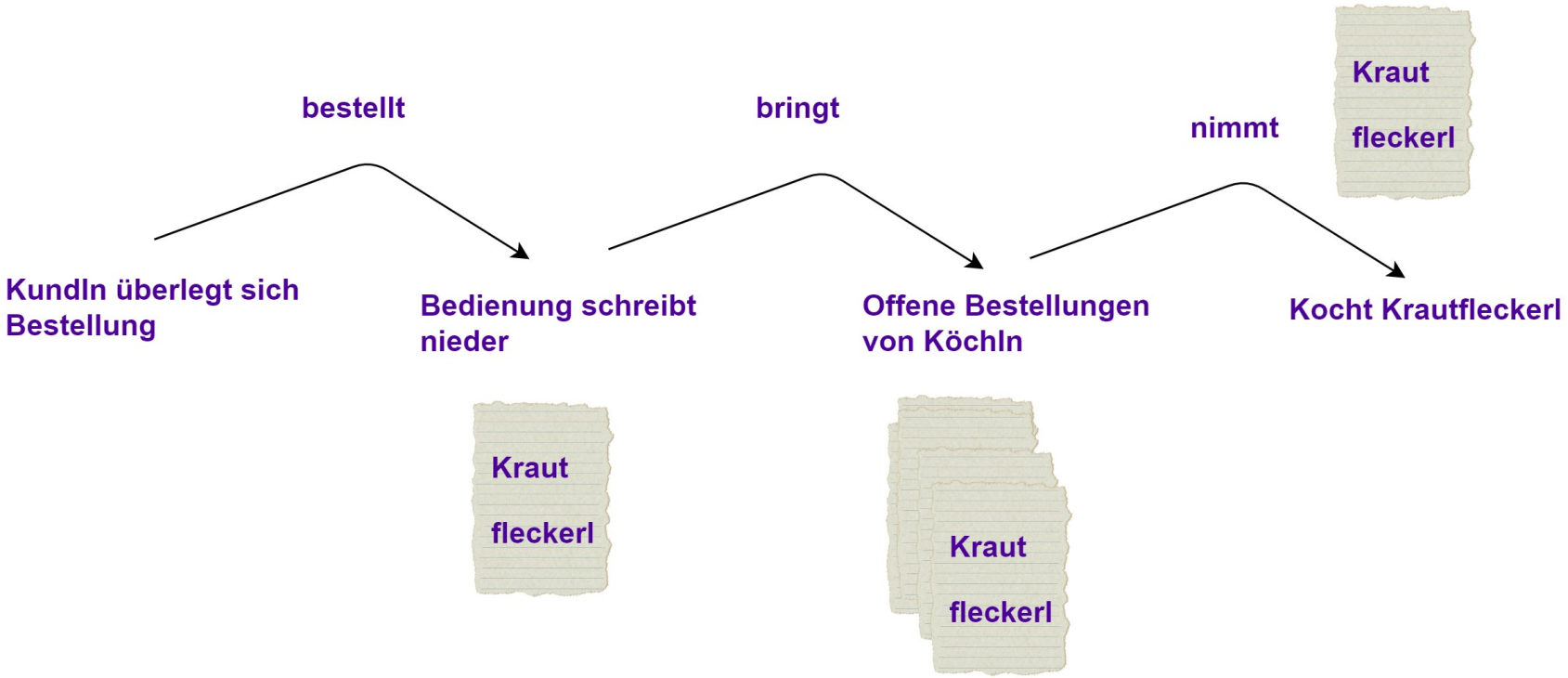
SMARTE FERNBEDIENUNG

(NICHT GANZ SO) SMARTE FERNBEDIENUNG

```
public void taste1() {
    tv.ein();
}
public void taste2() {
    tv.aus();
}

public void taste3() {
    musik.ein();
}
public void taste4() {
    licht.ein();
}
```

METAPHER BESTELLUNG



METAPHER BESTELLUNG

- Bestellung kapselt Auftrag Essen vorzubereiten
- Bestellung als Objekt mit dem Auftrag Essen vorzubereiten

-> Separieren Erstellung des Auftrags von Empfänger/Ausführer des Auftrags

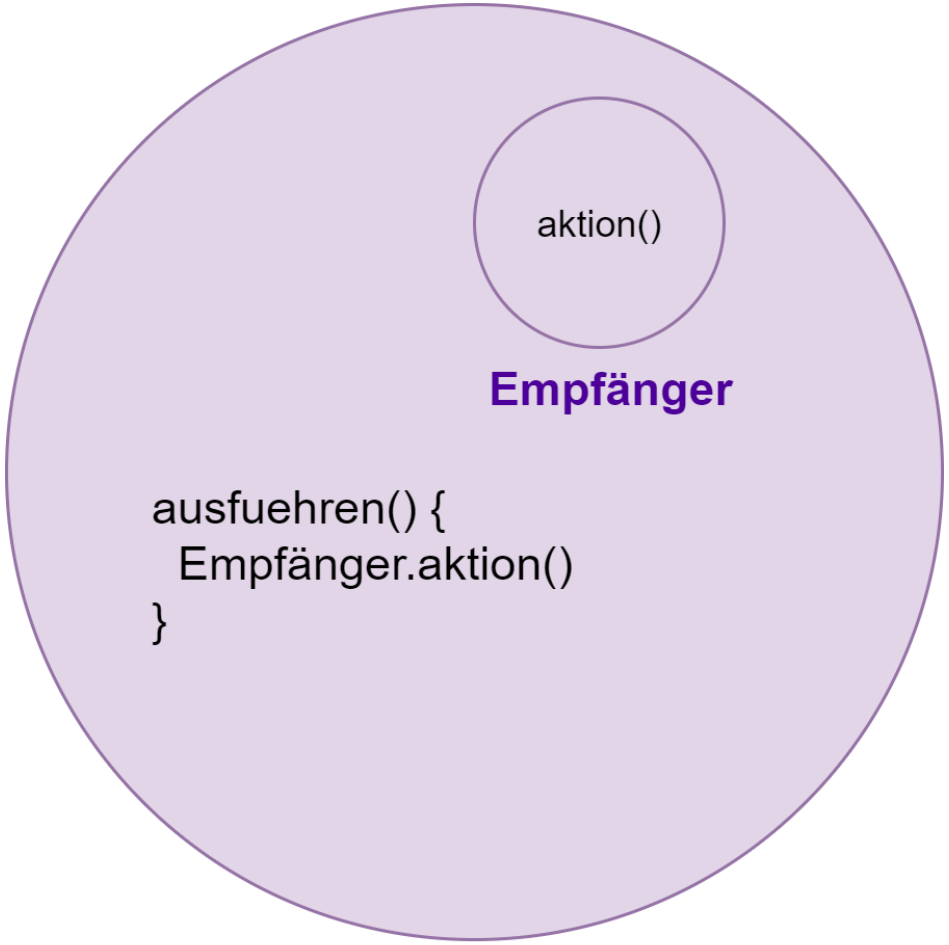
COMMAND PATTERN

- Setzen wir unser erstes Kommando um
- Let‘s code :)

COMMAND PATTERN

Das Command Pattern kapselt einen Befehl als ein Objekt. Damit kann man sie in eine Warteschlange stellen, loggen, und Operationen rückgängig machen.

KAPSELUNG COMMAND PATTERN

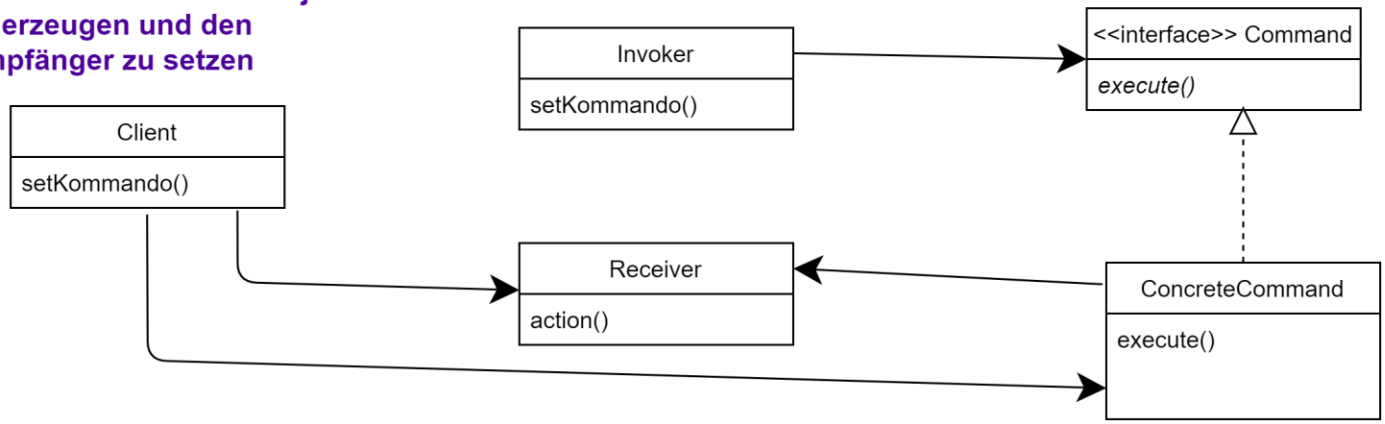


COMMAND PATTERN

Client verantwortlich das konkrete Kommando Objekt zu erzeugen und den Empfänger zu setzen

Ausführer hält ein Kommando und bittet Kommando zu gegebener Zeit seine execute Methode auszuführen

Interface für alle Kommandos



Receiver weiß wie Befehl umgesetzt werden muss (jede Klasse kann Receiver sein)

Konkretes Kommando bindet eine Aktion an einen Empfänger

Ausführer ruft execute Methode auf ---> Aktionen des Empfängers werden ausgeführt um Befehl zu erfüllen

BEISPIEL COMMAND

Erstellen Sie eine Klasse TV. Ein TV kann ein- oder ausgeschaltet sein und zwischen 40 Kanälen wechseln. Erstellen sie getter, setter für den Kanal bzw. die Möglichkeit den Kanal zu erhöhen bzw. zu erniedrigen. Erstellen Sie eine on und off Methode um den Fernseher ein- bzw. auszuschalten bzw. eine Methode um den Zustand abzufragen. (weiter auf nächster Seite)

BEISPIEL COMMAND (2)

- Nächster Schritt - Erstellen Sie folgende Command Klassen:
- Kanal auf bzw. ab um Kanäle zwischen 1 und 40 zu wählen
  - Kanalwahl schaltet Fernseher ein falls aus (weiter auf nächster Seite)

BEISPIEL COMMAND (3)

Testen sie die neuen Kommandos mit einer Fernbedienung die mehrere Knöpfe hat und für jeden Knopf ein anderes Kommando ausführen kann. Verwenden Sie dazu ein Array oder eine Map.

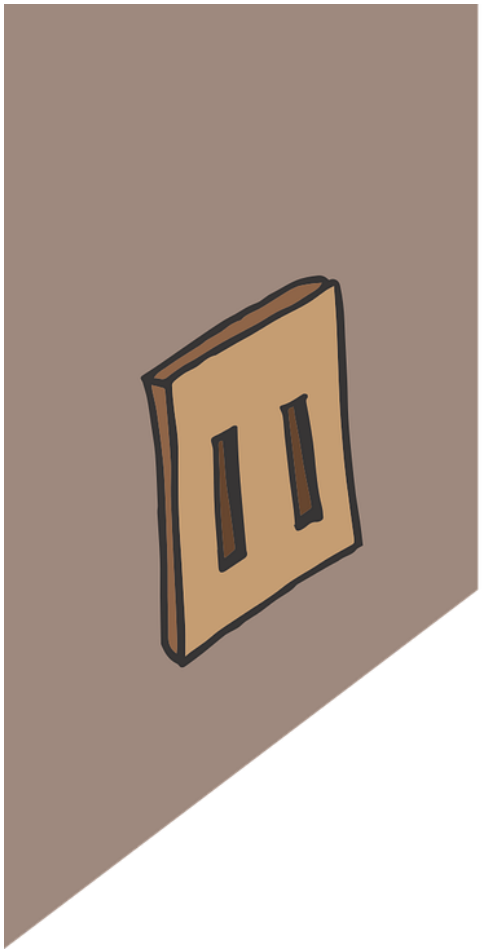
Schreiben Sie einen einfachen Eingabescanner mit dem sie die Buttons durch Eingabe der Nummer oder von Buchstaben „drücken“ können. (Solange bis sie z.B. „q“ eingeben)

Let‘s code Teil 2 ;)

BEISPIEL COMMAND TEIL 3

- Erweiterung undo Methode:
  - Implementieren Sie eine undo Methode für jedes konkrete Kommando.
  - Speichern Sie die getätigten Kommandos in einer dafür geeigneten Datenstruktur innerhalb der Fernbedienung.
  - Erweitern Sie die Fernbedienung um eine undo Funktion, mit der Sie vorher getätigte Kommandos rückgängig machen können.
  - Die Eingabe "undo" soll diese Funktion ausführen.

ADAPTER PATTERN

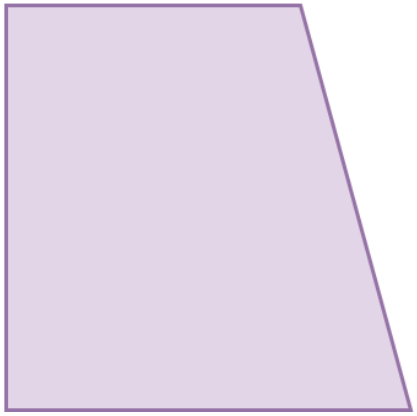


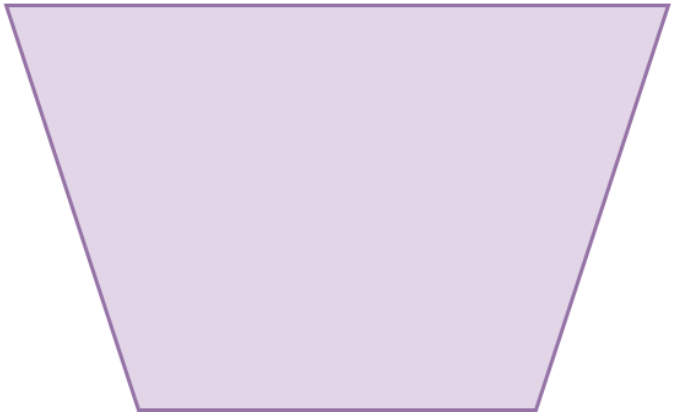
ADAPTER PATTERN

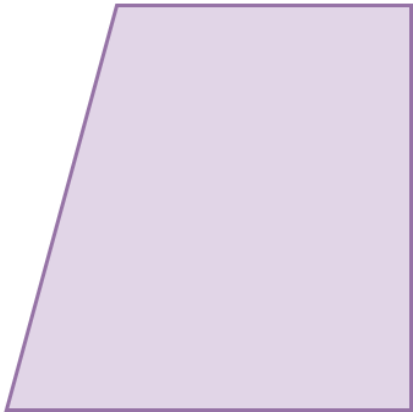
Bestehender Code

Adapter

Bibliothek







EIN TRUTHAHN KOMMT IN EINE BAR...

```
// Wir erinnern uns
public interface Duck {
    public void quack();
    public void fly();
}
// Von unserem amerikanischen Team
public interface Turkey() {
    public void gobble();
    public void fly();
}
```

EINSATZ ADAPTER

```
// Muss Zielinterface (Duck) umsetzen
public class TurkeyAdapter implements Duck
{
    Turkey turkey;
    public TurkeyAdapter(Turkey turkey)
    {
        this.turkey = turkey;
    }
    public void quack(){
        turkey.gobble();
    }
    public void fly() {
        // fliegt immer 2 kleine Schritte
        turkey.fly();
        turkey.fly();
    }
}
```

ADAPTER

Konvertiert das Interface einer Klasse in ein anderes Interface, welches der Client erwartet. Adapter lässt Klassen gemeinsam arbeiten, die sonst inkompatible Interfaces hätten.

ADAPTER BSP

- Beispielangabe Mediaplayer siehe Moodle