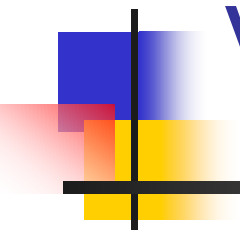
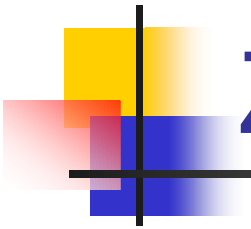


Verhaltensmuster

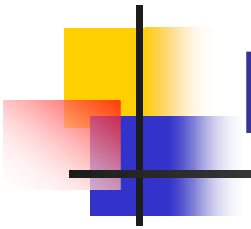
Befehl

(Command, Action)



Zweck

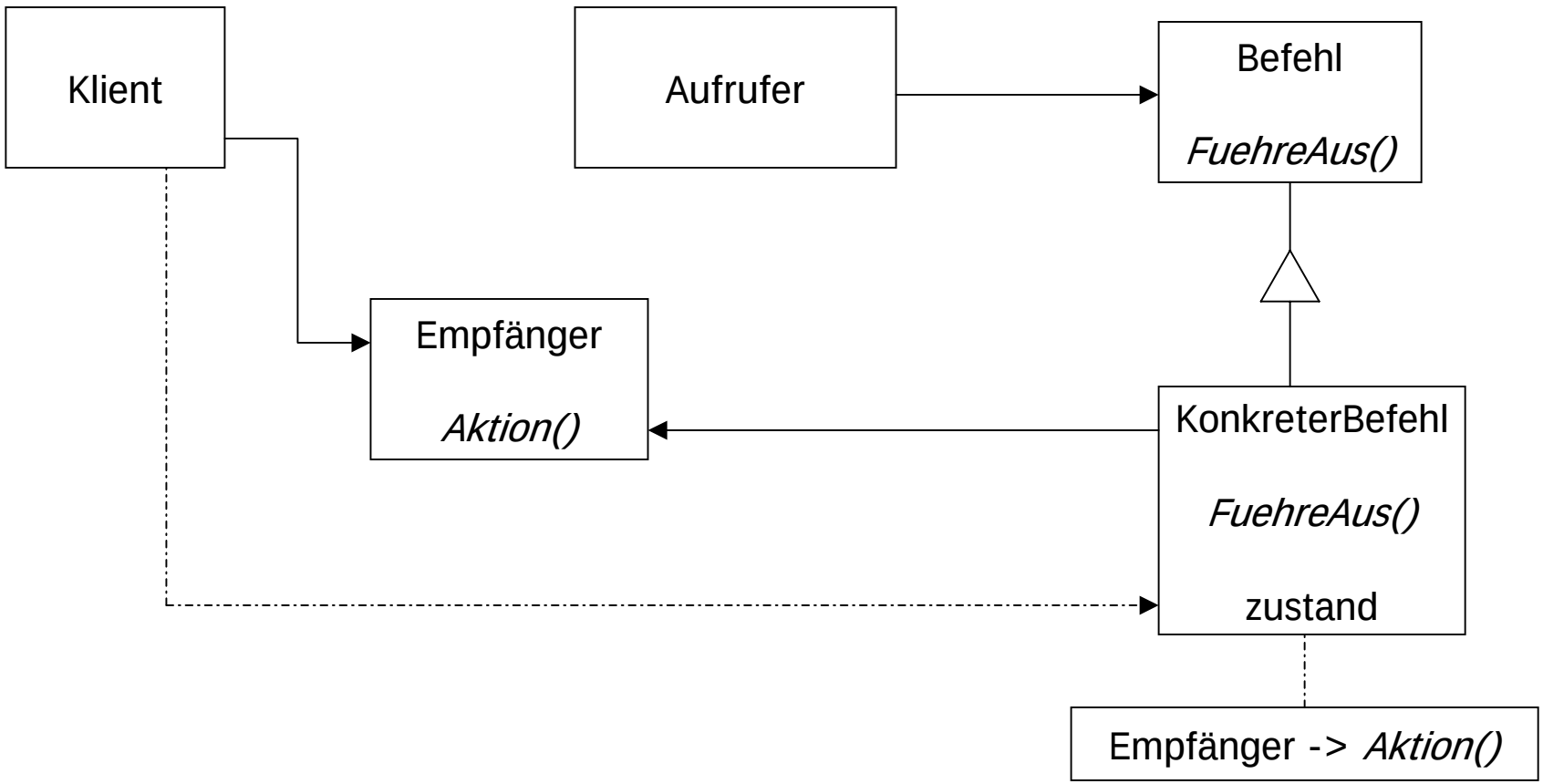
- **Kapsle einen Befehl als Objekt**
  - Klienten mit verschiedenen Anfragen parametrieren
  - OPs in Schlange stellen
  - Logbuch führen
  - OPs rückgängig machen

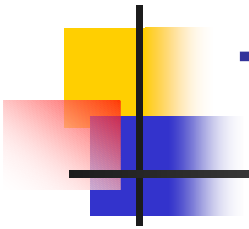


Motivation

- Klassenbibliothek enthalten z.B. Buttons und Menüs
- Klassenbibliothek implementiert zugehörige OPs nicht
- Anwendung weiß, wie Objekt behandelt werden kann
- Steuerungselemente einer Klassenbibliothek können Anfragen an unbekannte Anwendungsobjekte richten, indem Anfrage zu Objekt gemacht wird
- Abstrakte Klasse Befehl deklariert Schnittstelle zum Ausführen von OPs
- Einfachster Fall: nur abstrakte FuehreAus()-OP

Struktur

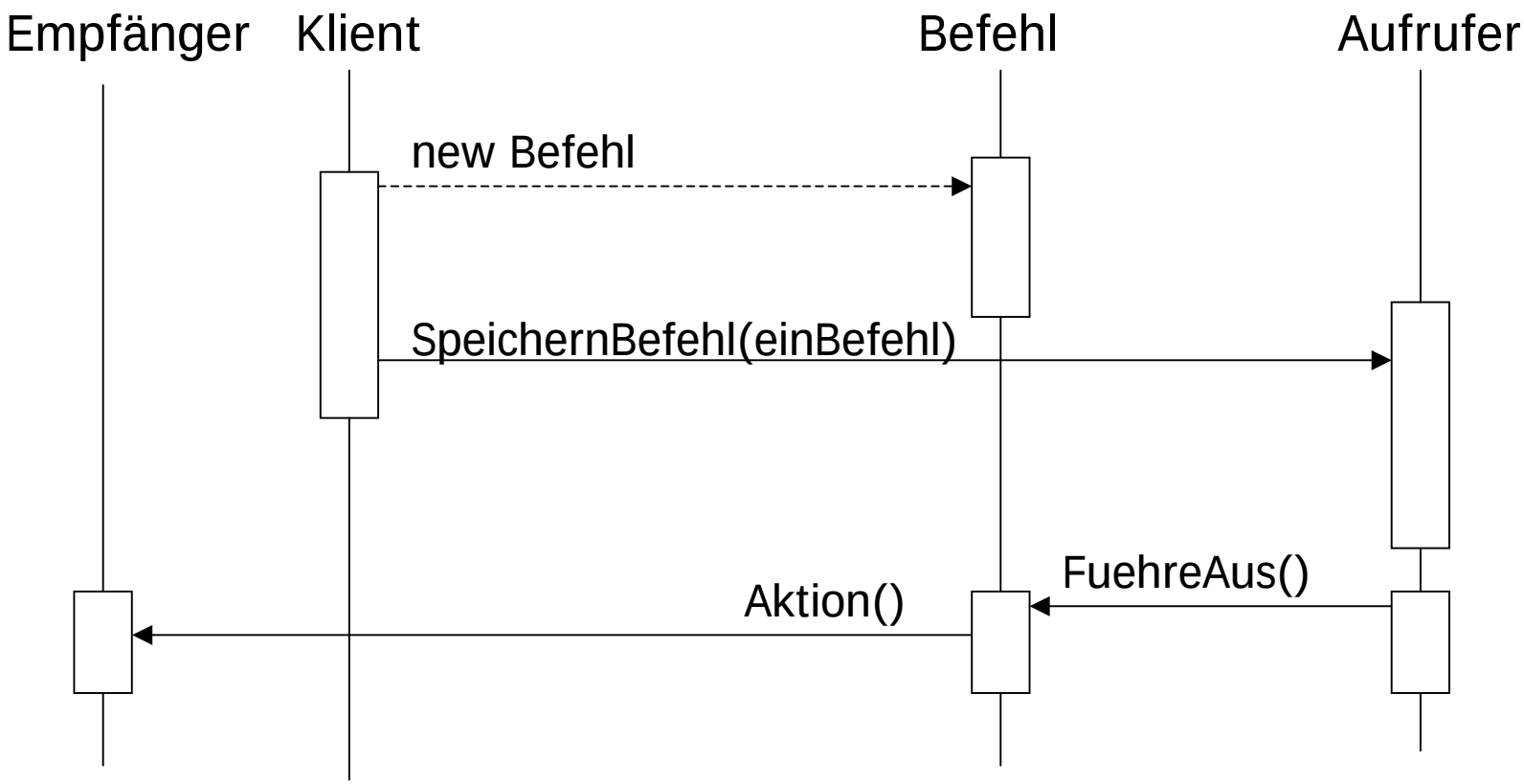

Klient
Aufrufer
Befehl
FuehreAus()
Empfänger
Aktion()
KonkreterBefehl
FuehreAus()
zustand
Empfänger -> Aktion()



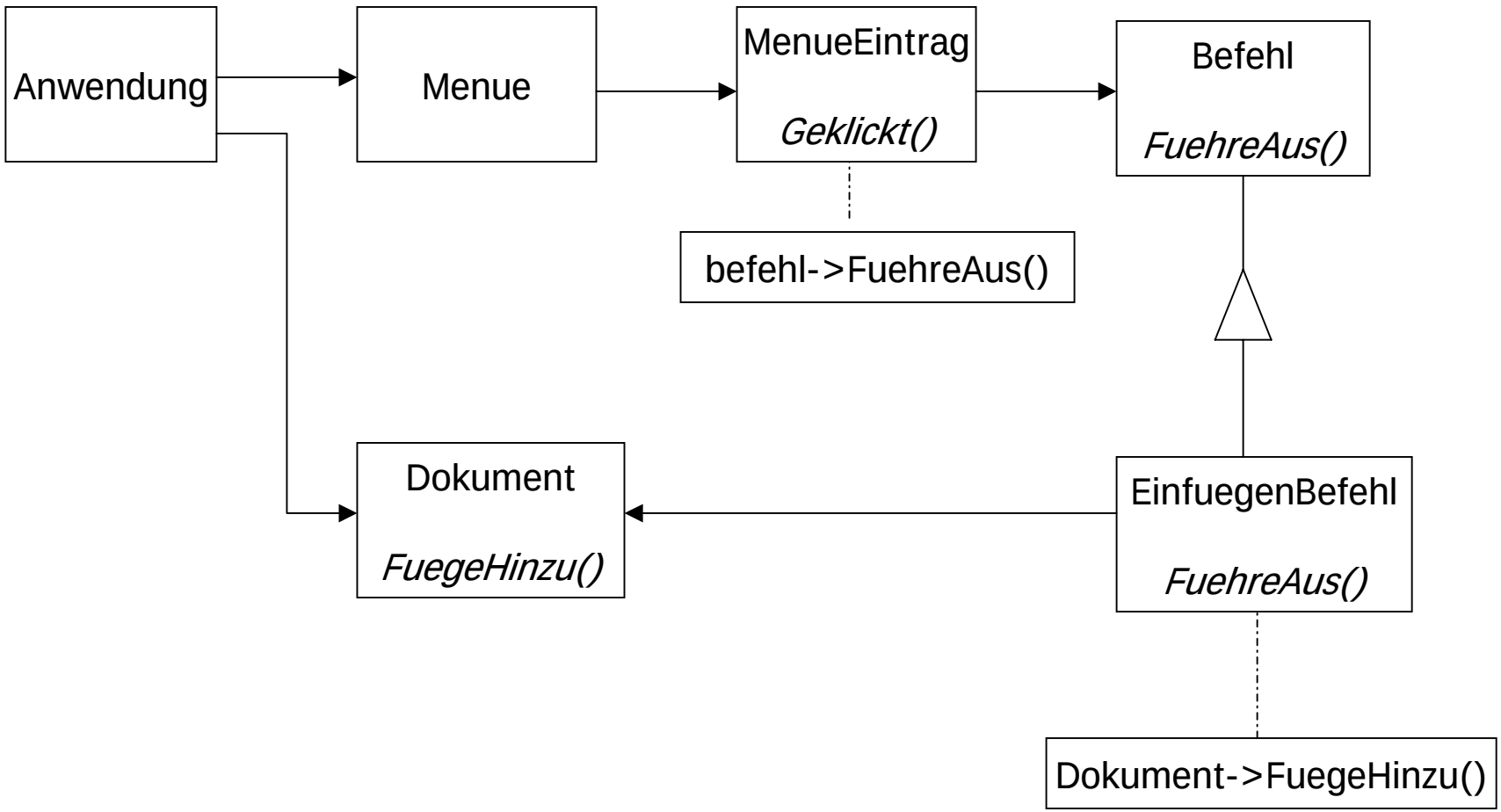
Teilnehmer

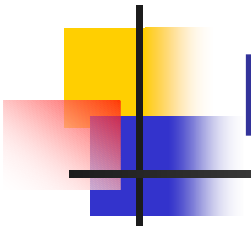
- Aufrufer / Auslöser (MenuEintrag)
  - befiehlt dem Befehlsobjekt, Anfrage auszuführen
- Befehl
  - deklariert eine Schnittstelle zum Ausführen einer OP
- Konkreter Befehl
  - definiert Anbindung eines Empfängers an eine Aktion
  - implementiert FuehreAus durch Aufrufen entsprechender OPs beim Empfänger
- Klient
  - erzeugt ein KonkreterBefehl-Objekt und übergibt diesem den Empfänger
- Empfänger
  - weiß, wie die OPs auszuführen sind

Interaktion



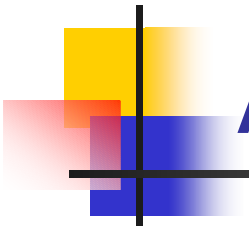
Befehlsmuster – Einfügen





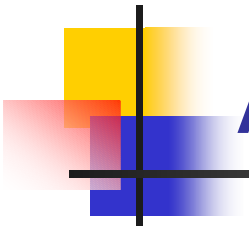
Konsequenzen

- Befehlsmuster entkoppelt das Anfrage-auslösende-Objekt von dem Anfrage-umsetzenden Objekt
- Befehlsobjekte können manipuliert und erweitert werden
- mehrere Befehlsobjekte können ein zusammengesetztes Befehlsobjekt bilden
- neue Befehlsobjekte können leicht hinzugefügt werden, da keine existierenden Klassen geändert werden müssen



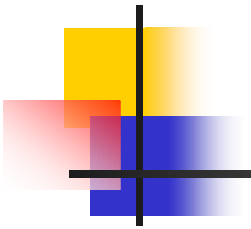
Anwendbarkeit (I)

- Objekte mit auszuführender Aktion parametrieren
  - z.B. MenueEintrag-Objekte
- Anfragen zu versch. Zeitpunkten spezifizieren, aufreihen und ausführen
  - unabhäng. Lebensdauer eines Befehlsobjektes
  - Befehlsobjekt zu anderen Prozess transferieren und ausführen
- Undo
  - Zustände in einer Befehlsgeschichte speichern
  - sequentieller Zugriff auf diese Liste
  - unbegrenztes Undo / Redo



Anwendbarkeit (II)

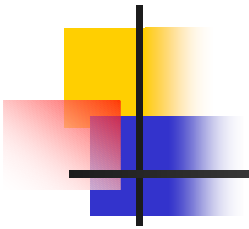
- Mitprotokollieren von Änderungen
  - Befehlsklassenschnittstelle erweitern (Laden und Speichern)
  - Logbuch dadurch anlegbar
  - nach Systemabsturz erneutes Ausführen der gespeicherten Befehlsobjekte
- System mittels komplexer OPs strukturieren, die aus primitiven OPs aufgebaut sind



Implementierung (I)

Intelligenz von Befehlsobjekten

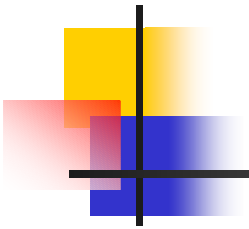
- Befehlsobjekte nur Verbindung zw. Empfänger und den OPs, die die Anfrage umsetzen
- Befehlsobjekte aber auch mit extrem umfangreichen Funktionen möglich, die alle Funktionalität selbst implementieren
  - Befehle von Klassen unabhängig
  - kein passender Empfänger existiert
  - Empfänger ist nur implizit bekannt



Implementierung (II)

Undo / Redo

- Konkrete Befehlsklasse um einen Zustand erweitern, dieser speichert: Empfänger, Argumente für die OPs, Originalwerte im Empfänger
- nur ein Undo
  - nur zuletzt ausgeführtes Befehlsobjekt speichern
- mehrfaches Undo
  - Befehlsgeschichte speichern, in dieser Liste sequentiell Vor- und Zurück-gehen ermöglicht Undo und Redo
  - maximale Listenlänge = Anzahl der Undo- und Redo-Ebenen



Implementierung (III)

Vermeidung von Fehlerlawinen im Undo

- Fehleransammlung, wenn Befehle mehrfach ausgeführt, rückgängig gemacht und erneut ausgeführt werden
- Verfremdung vom Originalzustand
- Lösung: weitere Zustandsinformationen im Befehlsobjekt speichern

Beispielcode Allgemein

- public interface **Befehl** {
  public void FuehreAus();
  }
- class **KonkreterBefehl** implements **Befehl** {
  public **KonkreterBefehl** (*Befehlsempfänger b*){
  // Befehlsempfänger und evtl. Parameter merken
  }
  public void **FuehreAus**() { *// Operation durch den*
  // Befehlsempfänger ausführen lassen
  }
  }

Beispielcode Einfügen

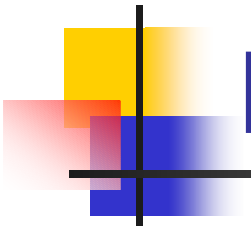
```
public interface Befehl {
        public void FuehreAus();
}
class EinfuegenBefehl implements Befehl {
        public EinfuegenBefehl (Dokument d ){
                doc = d;
        }
        public void FuehreAus() {
                doc.FuegeHinzu();
        }
        private Dokument doc;
}
```

Beispielcode Beenden

```
public interface Befehl {
        public void FuehreAus();
}
class BeendenBefehl implements Befehl {
        public BeendenBefehl (Anwendung a ){
                anw = a;
        }
        public void FuehreAus() {
                anw.exit();
        }
        private Anwendung anw;
}
```

Beispielcode Undo / Redo

- public interface **Befehl** {
  public void **FuehreAus()**;*// Operation ausführen*
  public void **Rueck()**; *// Operation wieder rückgängig machen*
  }



Beispielcode Undo / Redo

```
class VerschiebenBefehl implements Befehl {
    public VerschiebenBefehl(GraphObj o, int dx, int dy) {
        obj = o; deltax = dx; deltay = dy;
    }
    public void FuehreAus() {
        obj.move(deltax, deltay);
    }
    public void Rueck() {
        obj.move(-deltax, -deltay);
    }
    private GraphObj obj;
    private int deltax;
    private int deltay;
}
```