Zuständigkeitskette
(Chain of Responsibility)

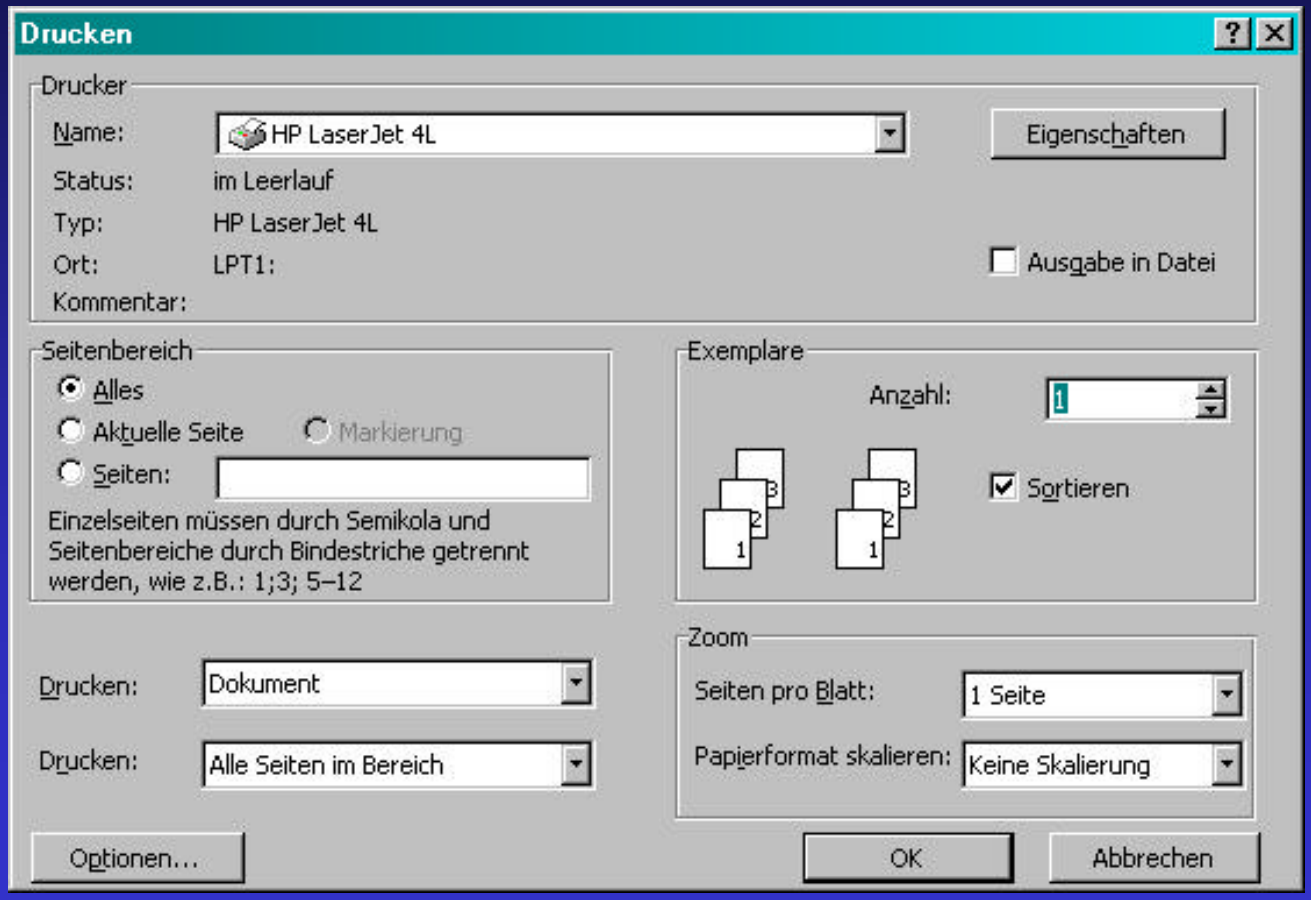
von Dirk Hain

Gliederung

1. Motivation
2. Grundidee und Realisierung
3. Anwendbarkeit
4. Struktur
5. Teilnehmer des Musters
6. Konsequenzen
7. Implementierungsalternativen
8. Verwandte Muster
9. Bekannte Anwendungen
10. Quellen

1. Motivation

Kontextabhängiges Hilfesystem



1. Motivation

- Hilfeinformationen hinter jedem Teil der GUI
- Jeder Teil hat spezifische Informationen (abhängig vom derzeitigen Kontext)

- Keine spezifischen Informationen vorhanden ⇨ allgemeinere Informationen anzeigen

1. Motivation

- Natürliche Organisation der Hilfe nach Grad der Allgemeinheit

  spezifisch  allgemein

- Anfragebearbeitung in dieser Reihenfolge
- „Weiterreichen“ der Hilfeanfrage

1. Motivation

Probleme:

- Verschiedene Objekte als Bearbeiter einer Anfrage möglich
- Bei Versandt unklar, wer Anfrage bearbeiten wird
- Dynamische Änderung der Bearbeitermenge erforderlich (Kontext)

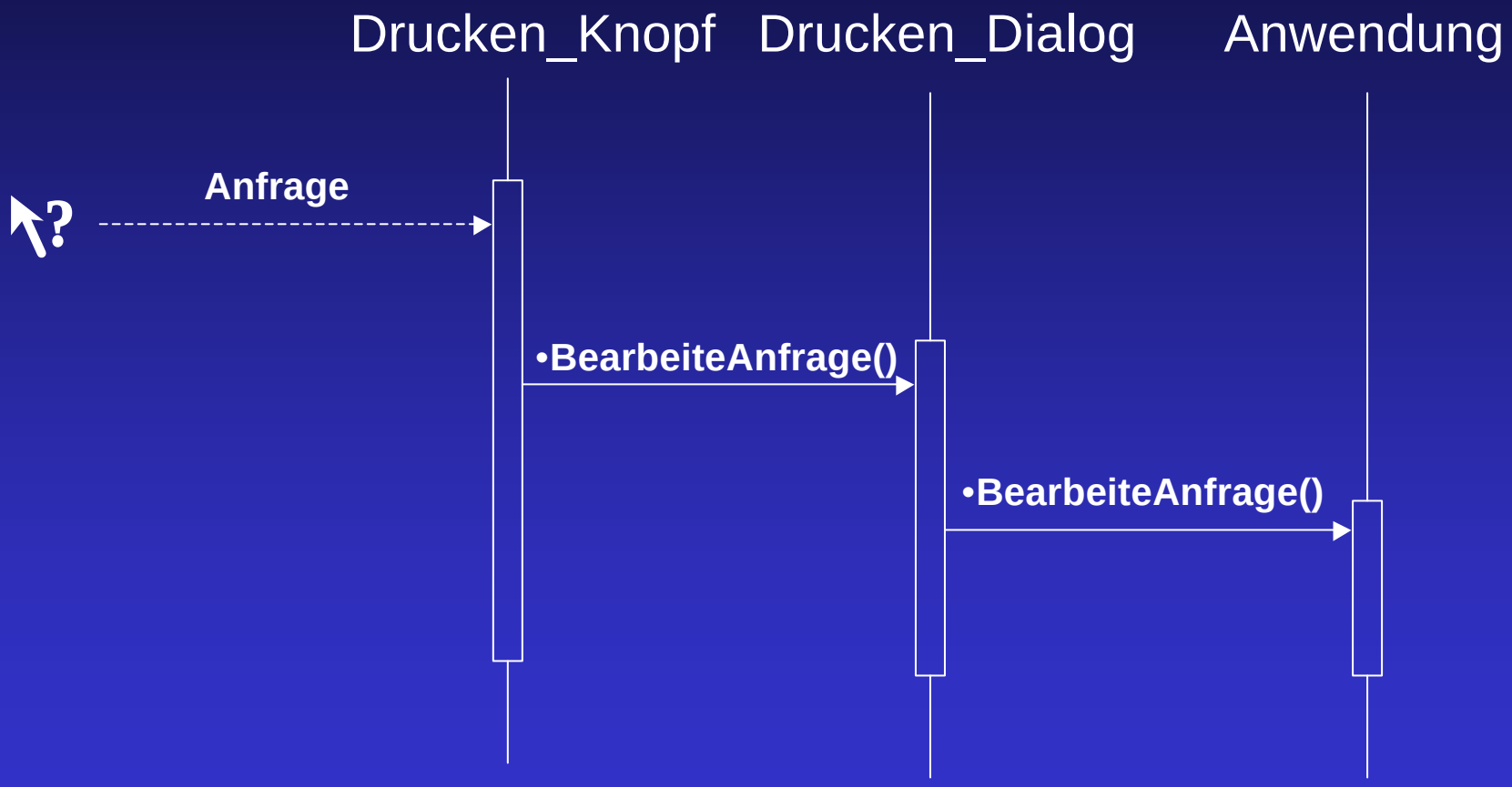
Lösung : ***Zuständigkeitskette***

2. Grundidee und Realisierung

- Grundidee: Entkopplung von Sender und Empfänger

- Anfragebearbeitung durch mehrere Objekte möglich
- Anfrageweiterleitung innerhalb dieser Objekte

2. Grundidee und Realisierung



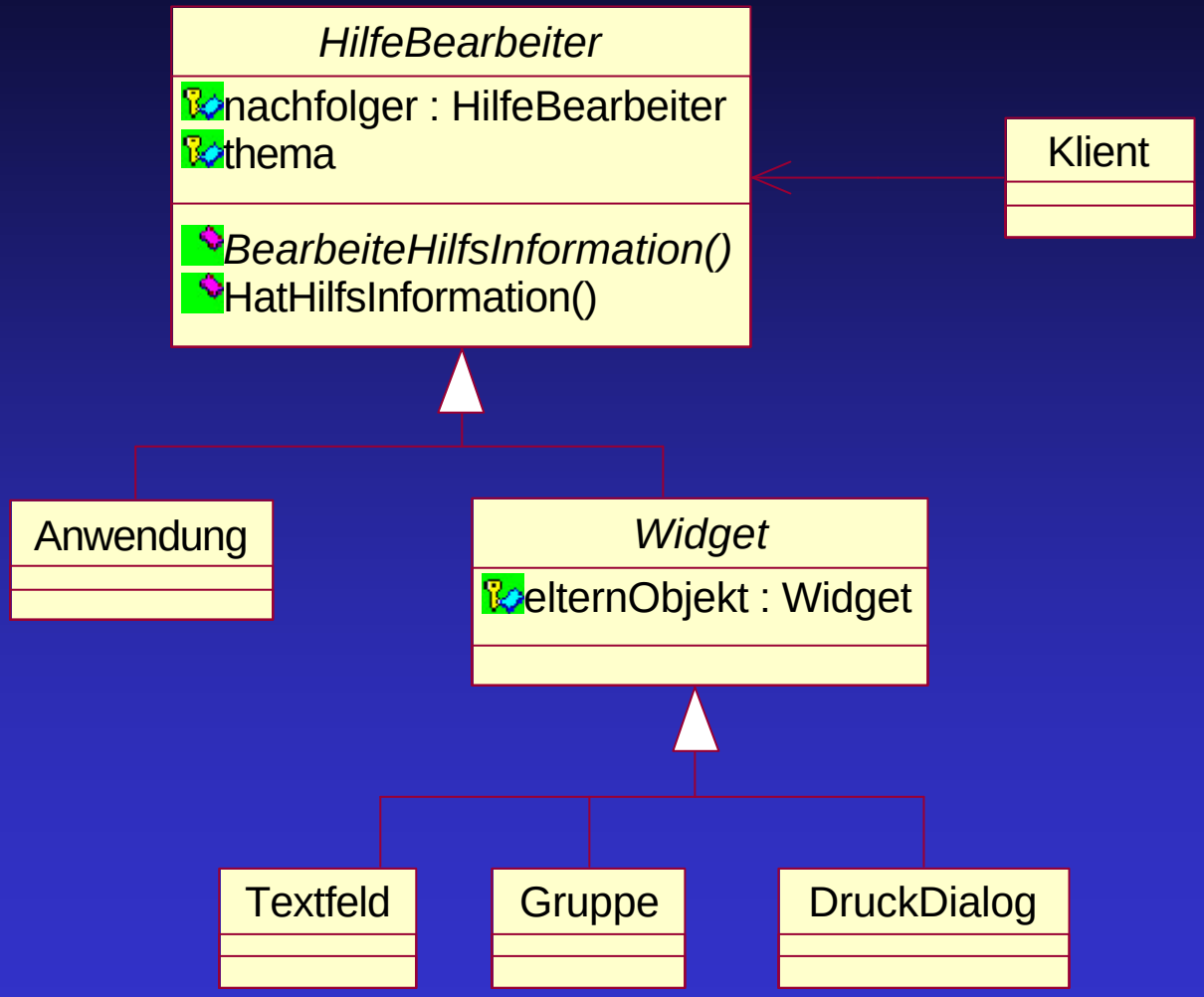
2. Grundidee und Realisierung

- Weiterleitung über definierte Schnittstelle
- ***Nachfolgeobjekt* muss bekannt sein**
- Objekt bearbeitet Anfrage oder leitet sie weiter

⇨ Definition einer abstrakten Oberklasse

⇨ Elemente des Hilfesystems von dieser abgeleitet

2. Grundidee und Realisierung

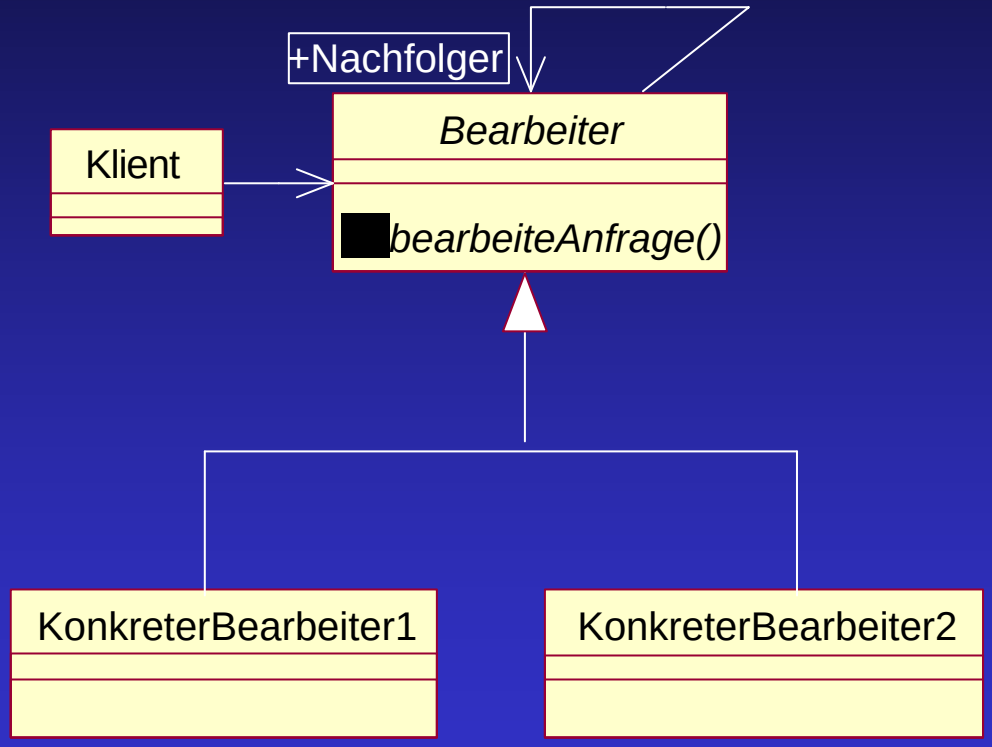


3. Anwendbarkeit

Folgende Bedingungen erfüllt das Muster:

- Bearbeitung durch mehr als ein Objekt möglich
- Anfrage ungerichtet (impliziter Empfänger)
- Dynamische Bearbeitermenge

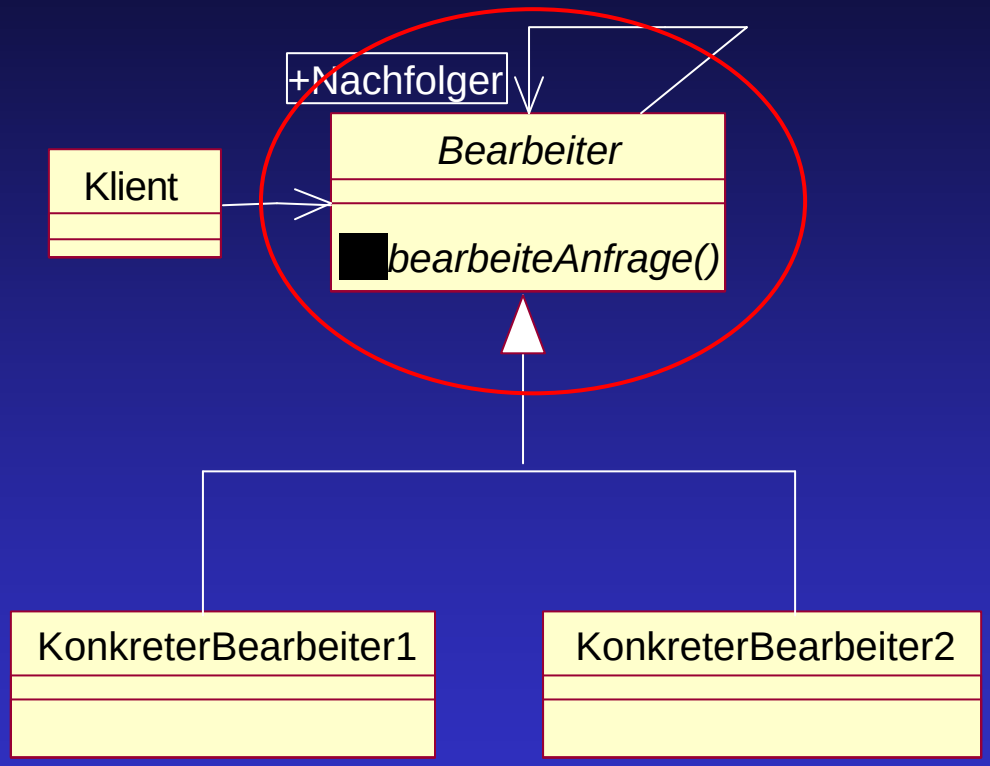
4. Struktur


+Nachfolger
Klient
Bearbeiter
bearbeiteAnfrage()
KonkreterBearbeiter1
KonkreterBearbeiter2

5. Teilnehmer des Musters

Bearbeiter :

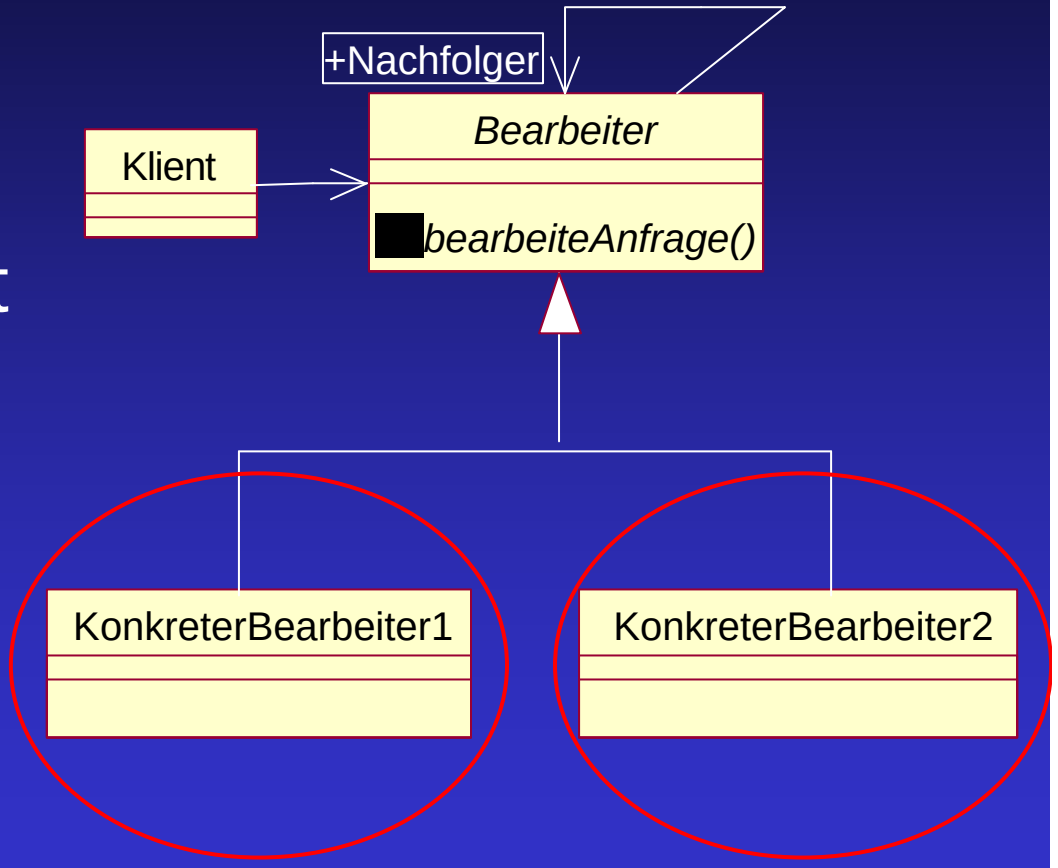
- Schnittstelle zur Anfragebearbeitung
- Referenz auf das Nachfolgeobjekt (optional)



5. Teilnehmer des Musters

KonkreterBearbeiter :

- Zugriff auf Nachfolger-Objekt
- Bearbeitung der Anfrage
- Ansonsten Weiterleitung an Nachfolger



5. Teilnehmer des Musters

Interaktion:

- Kette von Bearbeitern
- Klient stellt Anfrage
- Bearbeiter behandelt Anfrage
- Ansonsten Weiterleitung an nächstes Objekt

6. Konsequenzen

Reduzierte Kopplung (+)

- Vereinfachte Objektstruktur
- Nur ***Nachfolger***-Referenz

Hohe Flexibilität (+)

- Leichtes Einfügen (Entfernen) neuer Zuständigkeiten

Keine Abarbeitungsgarantie (-)

- Anfrage kann verloren gehen

7. Implementierungsalternativen (1/3)

Implementierung der Kette:

a) Nutzung existierender Verbindungen
   (Kompositum ⇨ Elternreferenzen)

b) Definition neuer Verbindungen
   ⇨ mehr Speicherplatz

7. Implementierungsalternativen (2/3)

Implementierung des Nachfolgers:

- Implementierung des Nachfolgers bereits in abstrakter Oberklasse (HilfeBearbeiter)
- Defaultimplementierung für *BearbeiteAnfrage() jetzt möglich*
- Defaultverhalten: Weiterleitung

7. Implementierungsalternativen (3/3)

Repräsentation von Anfragen

- Bisher: nur in *Bearbeiter* definierte Anfragen möglich
- Flexibler: Definition von Anfrageklassen

  ⟹ Bearbeitung je nach Typ

- Implementierung neuer Anfragetypen leicht (Ableitung)
- Kapselung der Anfrageparameter
- Aber : Typumwandlung zur Laufzeit !

8. Verwandte Muster

- Oft Anwendung mit *Kompositum*-Muster
- Kompositum definiert Elternobjektreferenz (optional)
- Dient als *Nachfolger* - Objekt

9. Bekannte Anwendungen

- In Klassenbibliotheken vor allem beim *EventHandling*
- *z.B. MacApp, ET++ aber auch Java 1.0*
- *Unidraw* – Framework für grafische Editoren

10. Quellen

- „Entwurfsmuster“ GoF
- Dirk Römer, „Verhaltensmuster“ (pdf der Uni Münster)
- „Entwurfsmuster“ (pdf der Uni Marburg)