

---

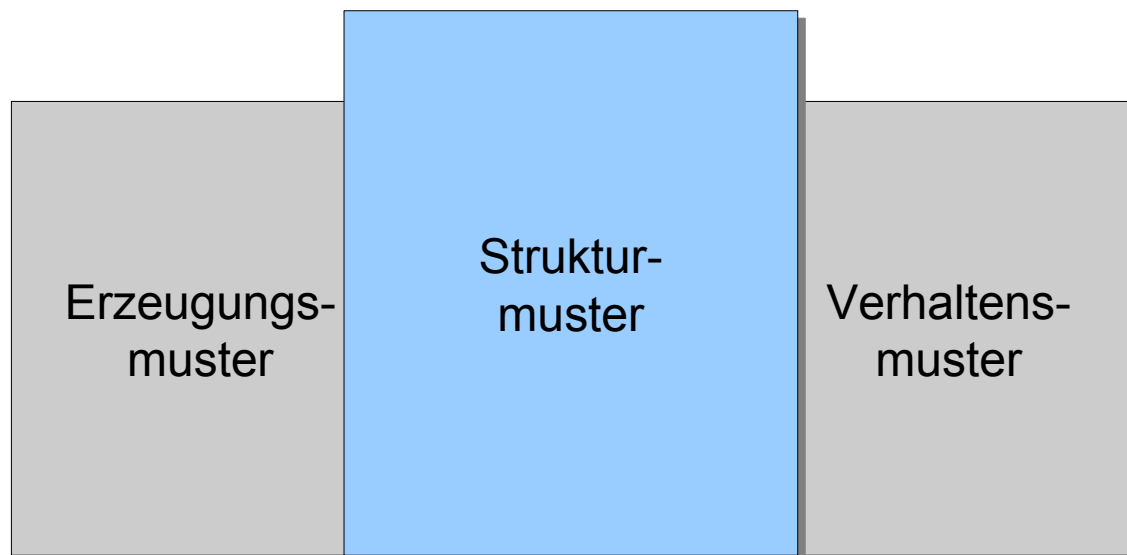
# Entwurfsmuster Der Dekorierer

Funktionskleidung für Objekte

# Der Dekorierer

---

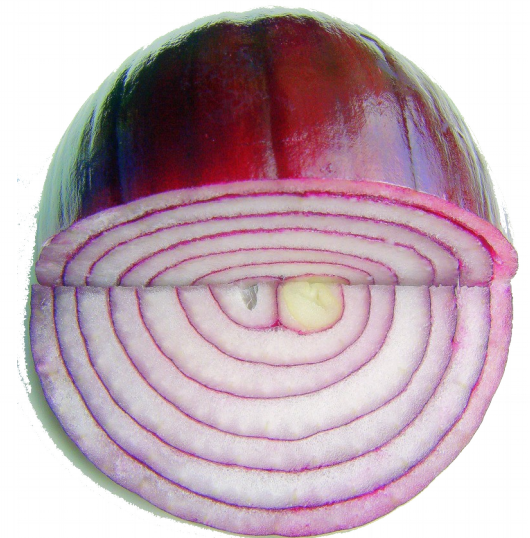
- Klassifikation
  - objektbasiertes Strukturmuster
  - Leichtgewichtig
  - Instanzenreich
- Alternativname: Decorator, Wrapper



# Zweck

---

- Dynamische Erweiterung eines Objekts um Zuständigkeiten
  - Schnittstellenänderung zur Laufzeit
- Funktionalitätsänderung ohne Unterklassenbildung

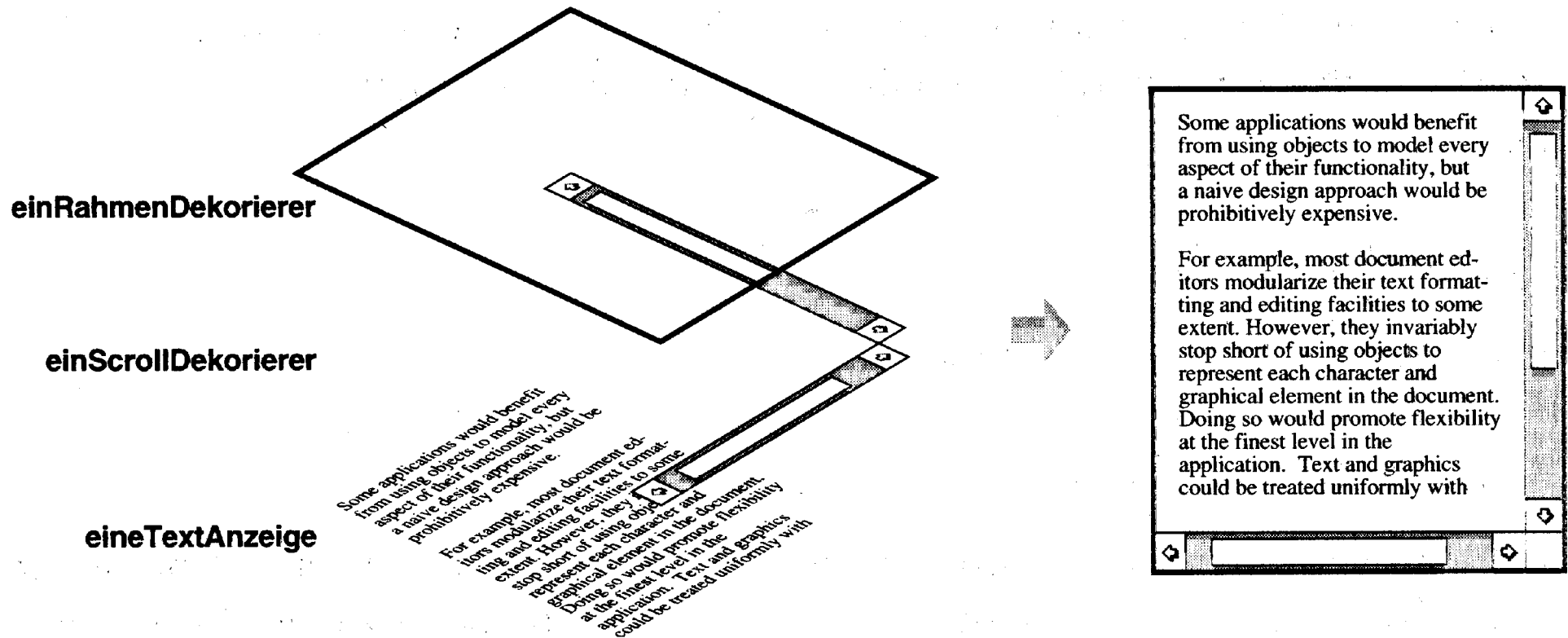


# Motivation

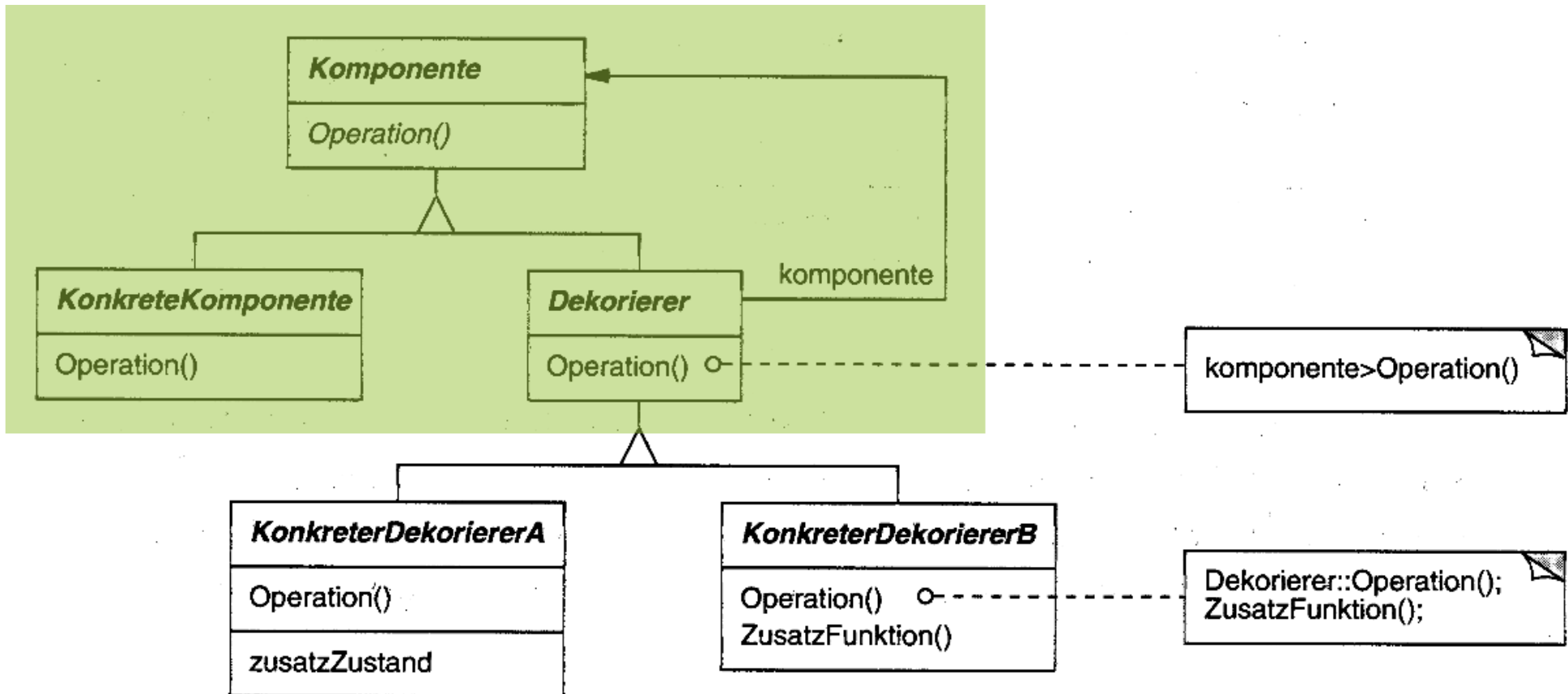
---

- Änderung der Funktionalität einzelner Objekte, ohne Klasse zu ändern
  - Beispiel: Textfeld, das bei Bedarf Scrollen soll
  - Vererbung funktioniert nur statisch
  - Klasse legt Zugehörigkeit und Funktionalität fest
    - Zur Laufzeit keine Änderung mehr
- Dekorierer
  - Umschliesst das Ursprungsobjekt
  - Fügt die zusätzliche Funktionalität hinzu

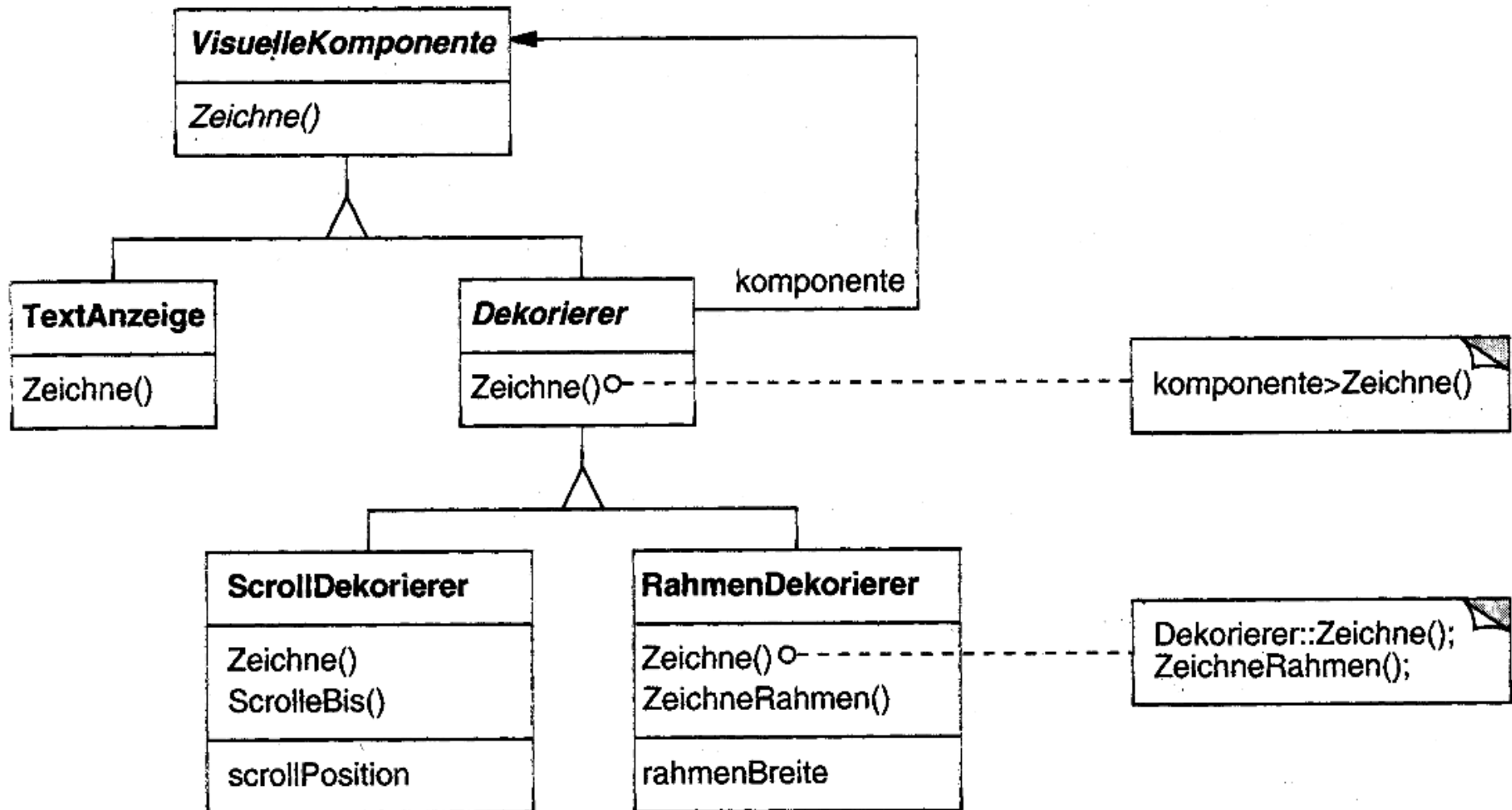
# Motivation im Bild



# Struktur



# Beispielstruktur



# Anwendbarkeit

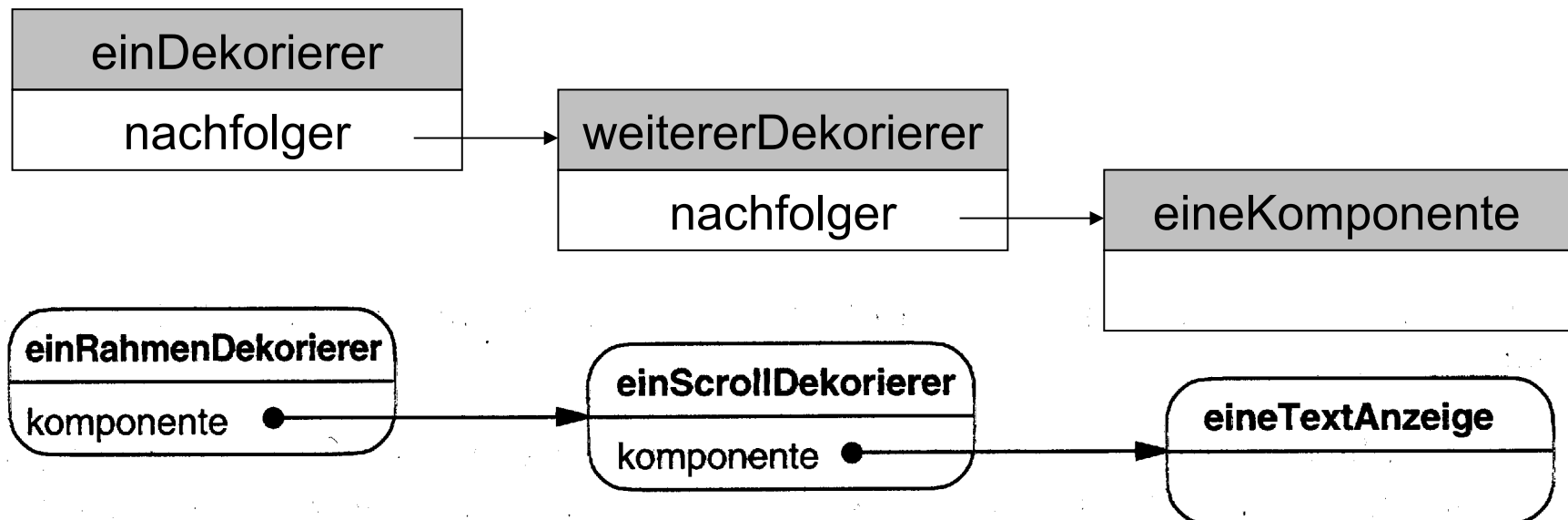
---

- Hinzufügen von zusätzlicher Funktionalität zu einzelnen Objekten
  - Dynamisch und transparent
- Funktionalität sollte wieder entfernt werden können
- Unterklassenbildung wäre unpraktisch
  - Grosse Anzahl voneinander unabhängiger Funktionalitätserweiterungen
    - Riesige Unterklassenmenge
  - Eventuell ist Oberklasse nicht ableitbar



# Interaktion

- Jeder Dekorierer leitet Anfragen an sein Komponentenobjekt weiter
  - Optional vor und nach dem Weiterleiten weitere Operationen durchführbar



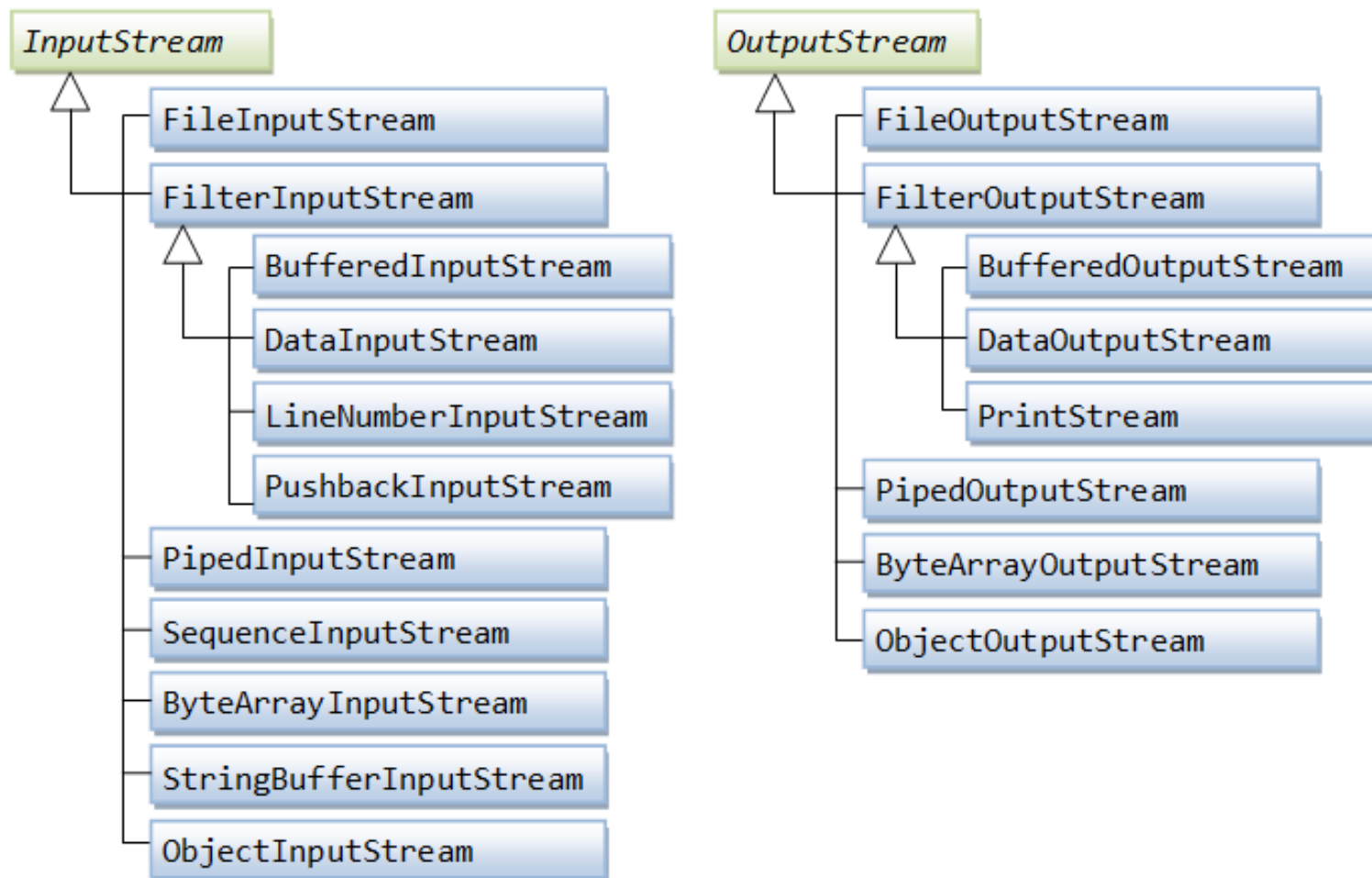
# Vorteile

---

- Größere Flexibilität im Vergleich zu statischer Vererbung
  - Funktionalität zur Laufzeit hinzufügbare und auch wieder entfernbare
  - Funktionalität ohne Probleme mehrfach hinzufügbare
- Vermeidet „Eierlegende Wollmilchsau“-Klassen
  - Funktionalität wird im Bedarfsfalle hinzugefügt
  - Einfache Klassen werden inkrementell mächtiger

# Beispiel-Implementierung

## Java Stream/Writer-Architektur



# Beispiel-Implementierung

## Java Stream/Writer-Architektur

```
final InputStream input =  
    new FileInputStream(dataFile);
```

```
final InputStream input =  
    new BufferedInputStream(  
        new FileInputStream(dataFile));
```

```
final InputStream input =  
    newLineNumberInputStream(  
        new BufferedInputStream(  
            new FileInputStream(dataFile)));
```

```
final InputStream input =  
    newDecodingStream(anotherSecretKey,  
        newDecodingStream(secretKey,  
            newUnzippingStream(  
                newObjectInputStream(  
                    newLineNumberInputStream(  
                        newBufferedInputStream(  
                            newFileInputStream(dataFile))))));
```

*InputStream*

FileInputStream

FilterInputStream

BufferedInputStream

DataInputStream

LineNumberInputStream

PushbackInputStream

PipedInputStream

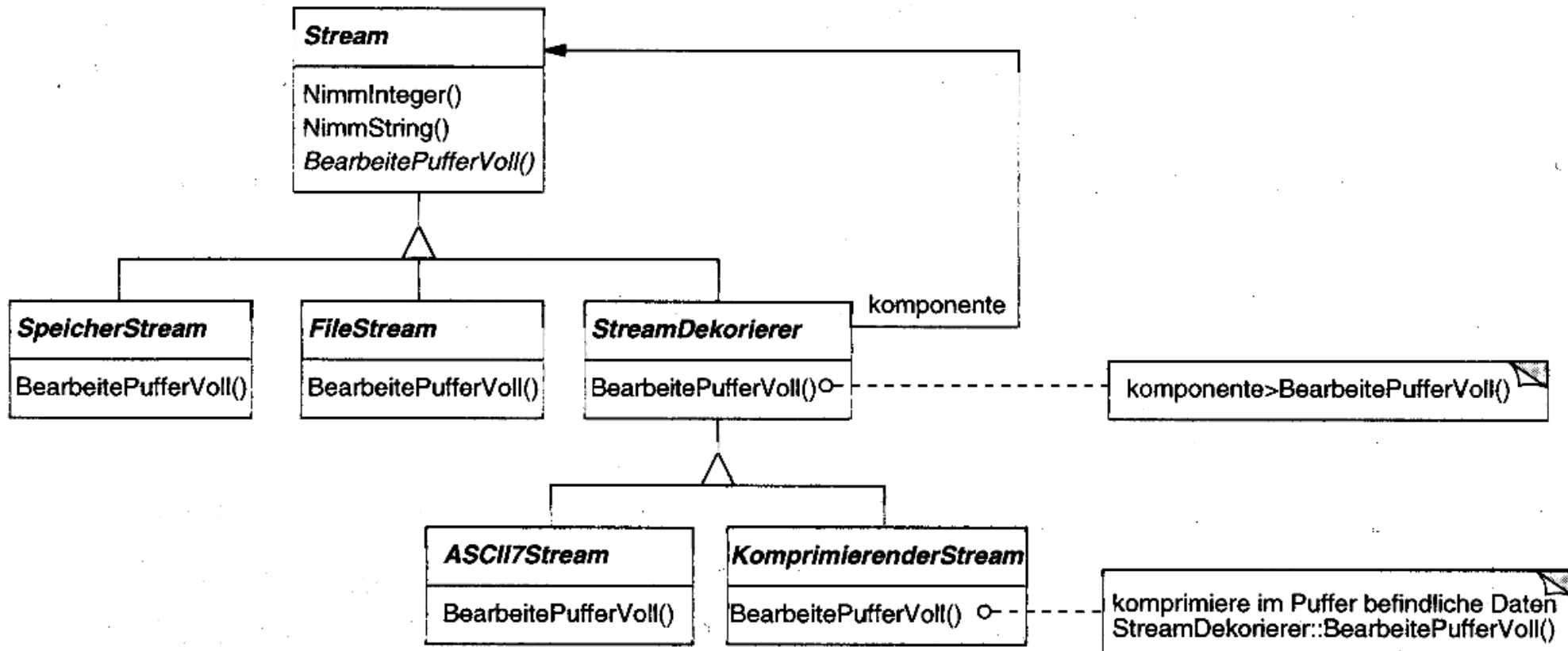
SequenceInputStream

ByteArrayInputStream

StringBufferInputStream

ObjectInputStream

# Struktur für dekorierte Streams



# Fortgeschrittene Implementierung

---

- Konstante Schnittstellen
  - Dekorierer benötigen jeweils gleiche Grundschnittstelle
  - Zusatzmethoden und –daten sind kein Problem, allerdings nicht automatisch sichtbar
  - Oft wird deshalb mit Ability-Interfaces (z.B. „Scrollable“) gearbeitet
  - Nachteil: Der Klient muss vorher abprüfen

# Nachteile

---

- Dekorierer und die Komponente sind nicht identisch
  - Dekorierer ist eine „durchsichtige Hülle“
  - Auf Objektidentität ist kein Verlass mehr
- Viele kleine Objekte
  - Dekorierer führen zu Systemen mit vielen kleinen oft gleichartig aussehenden Objekten
  - Unterscheidung der Verantwortlichkeiten liegt oft nur noch in den Verbindungen, die Klasse ist eher nebensächlich
  - Sehr schwer zu verstehen und zu debuggen

# Nachteile

---

- Funktionalitätserweiterungen sind flüchtig
  - Zu Systemstart müssen alle Erweiterungen wieder hinzugefügt werden
- Die Klasse eines Objekts wird relativ unbedeutend
  - Die tatsächlichen Möglichkeiten eines Objekts kommt aus den „Hüllen“



# Zusammenfassung

- Dekorierer
- Objektbasiertes Strukturmuster
- Dynamische Erweiterung der Funktionalität von Kernobjekten
- Die Verkettung einfacher Objekte ergibt komplexe Funktionalität
- „Favour Composition over Inheritance“
- Nachteil: Objektidentität und -typ verlieren an Bedeutung

