Dekorierer

Sebastian Walther
Falk Werner
Michael Ziese

HTWK-Leipzig
FbIMN 03IN-TI

1. Klassifizierung & Musternamе

- Klassifikation des Dekorierers
  - zählt zu den objektbasierenden Strukturmustern
  - Gültigkeitsbereich: kompositionsbasiert, d.h.
    - beschreiben von dynamischen/rekursiven Beziehungen, die zur Laufzeit geändert werden können
- Alias zu Dekorierer
  - Decorator
  - Gebundener Umwickler
  - Wrapper

2. Zweck eines Dekorierers

- Was macht dieses Muster?
  - erweitert Objekt dynamisch um Zuständigkeiten
  - Funktionalität einer Klasse aufrüsten
- Welches Problem löst es?
  - bietet flexible Alternative zur Unterklassenbildung
- Prinzip:
  - Zusammensetzung unterschiedl. Objekte mittels Objektkomposition, um zusätzl. Funktionalität der Objekte zu erreichen.

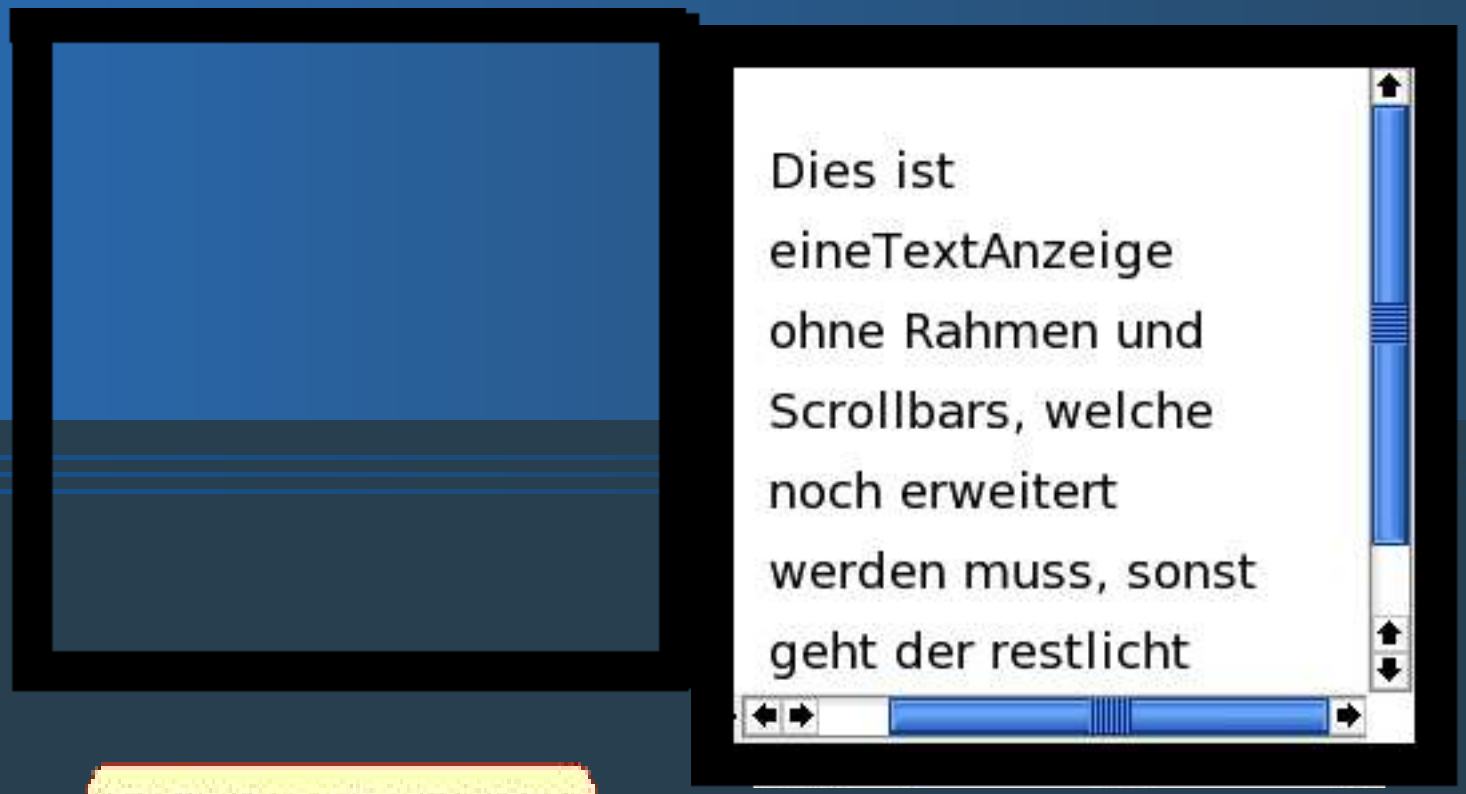
3. Motivation I

- Szenario: beliebige GUI- Komponente
  - mit Umrahmungen ausstatten
  - um Scroll- Funktion erweitern
- Lösungsansatz:
  - Ergänzen von Objekten um Funktionalitäten ohne ihre Klasse zu ändern
  - Einschließen der Komponente in einem anderen Objekt, dem **Dekorierer**
  - transparente Schnittstelle des Dekorierers

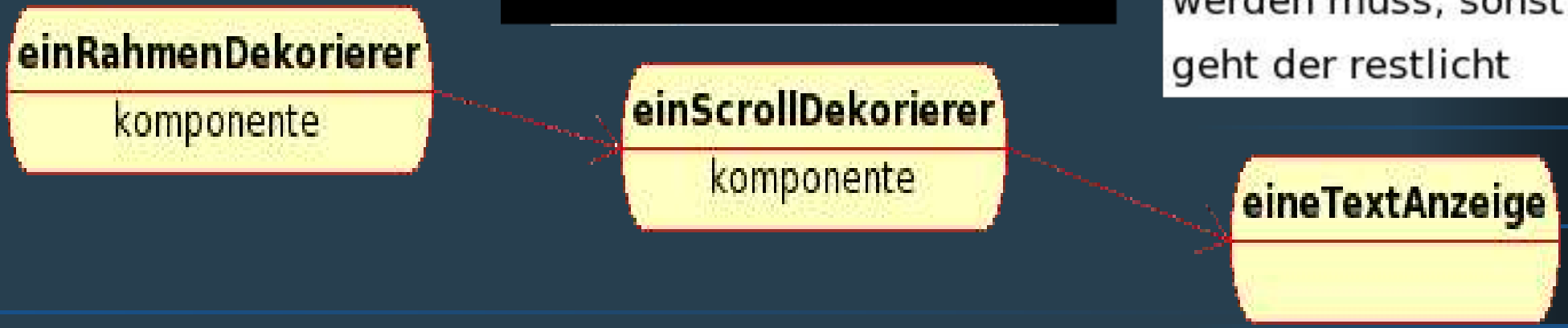
3. Motivation II

- Die umschließende Komponente
  - leitet Operationsaufrufe weiter an das eingeschlossene Objekt
  - führt vor oder nach Weiterleitung zusätzliche Operationen aus
  - lässt sich rekursiv schachteln
- Spezifisches Beispiel zum Szenario
  - TextAnzeigeObjekt ohne Rahmen und Scrollbars
  - Einsatz von Rahmen- und ScrollDekorierer

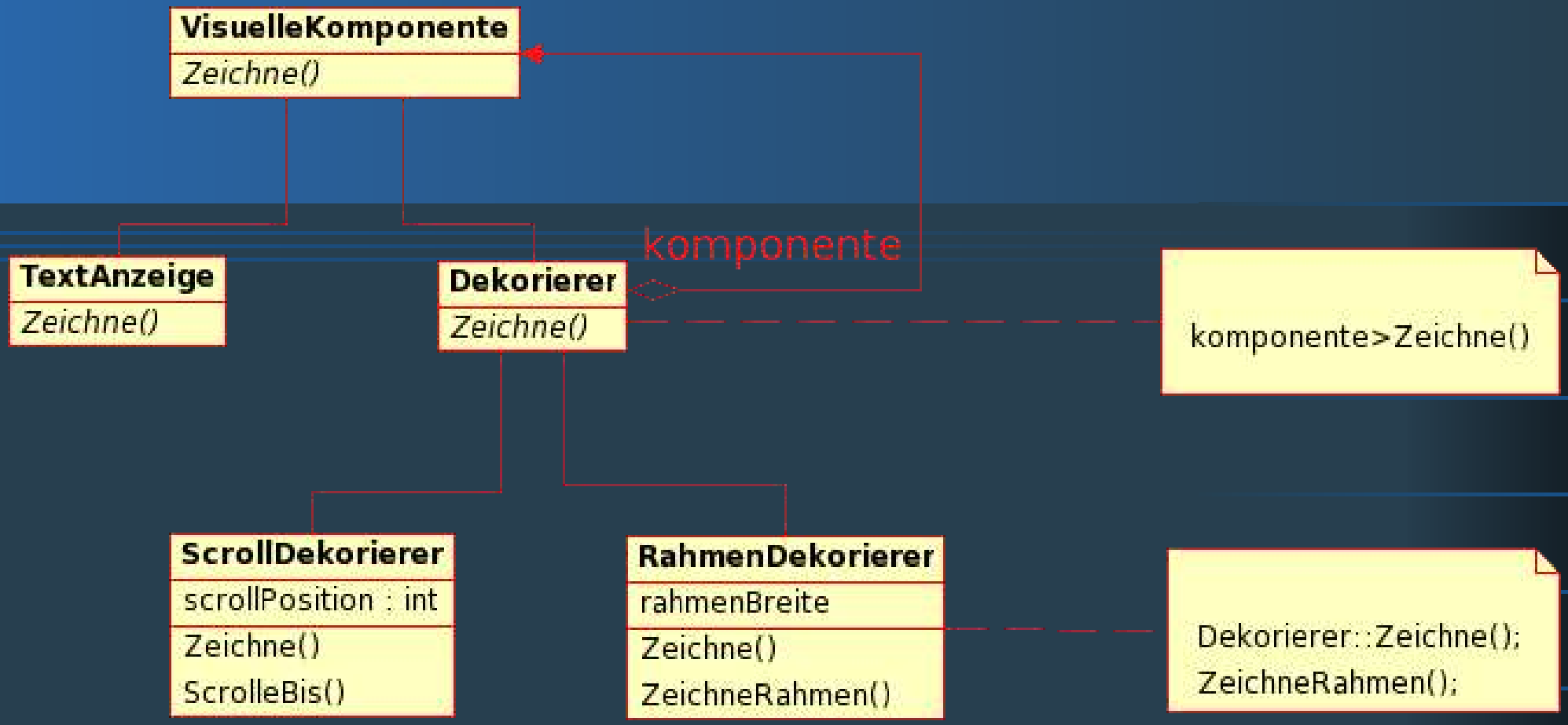
3. Motivation III



Dies ist
eineTextAnzeige
ohne Rahmen und
Scrollbars, welche
noch erweitert
werden muss, sonst
geht der restlicht



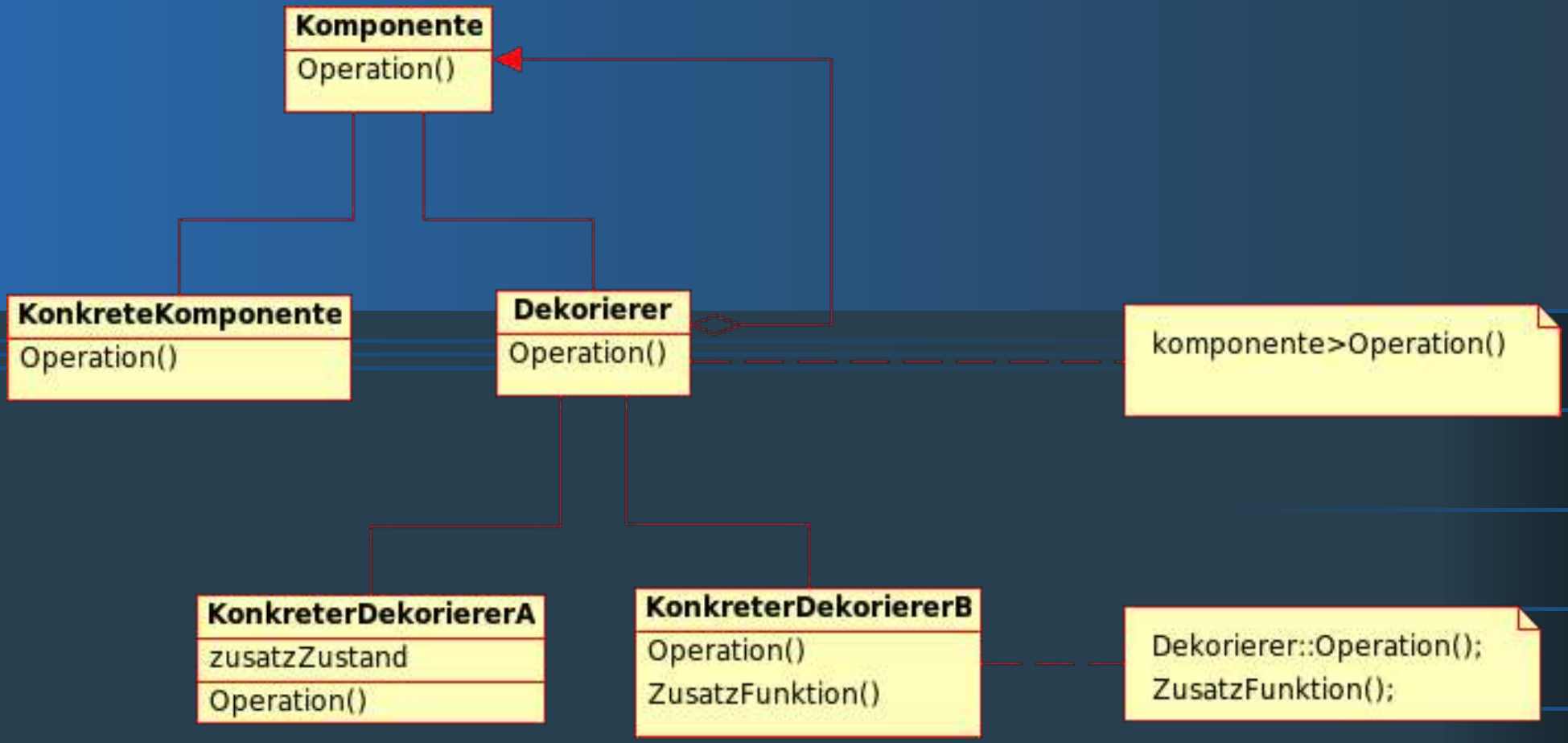
3. Motivation IV



4. Anwendbarkeit

- Wann wendet man das Dekoriermuster an?
  - um einzelnen Objekten zusätzliche Funktionalität dynamisch und transparent hinzuzufügen, ohne andere Objekte mit einzubeziehen
  - um Funktionalität hinzuzufügen, die auch wieder entfernt werden kann
  - wenn Klassen nicht ableitbar sind
  - Vermeidung explosionsartiger Vermehrung von Unterklassen

5. Struktur


Komponente
Operation()
KonkreteKomponente
Operation()
Dekorierer
Operation()
komponente>Operation()
KonkreterDekoriererA
zusatzZustand
Operation()
KonkreterDekoriererB
Operation()
ZusatzFunktion()
Dekorierer::Operation();
ZusatzFunktion();

6. Teilnehmer

- Komponente (z.B. VisuelleKomponente)
- KonkreteKomponente (TextAnzeige)
  - definiert Objekt, das um zusätzl. Funktionalität erweitert werden kann.
- Dekorierer
  - verwaltet eine Referenz auf ein Komponenten-objekt und definiert eine Schnittstelle, die der Schnittstelle von Komponente entspricht
- KonkreterDekorierer (z.B. ScrollDekorierer)
  - fügt der Komponente Funktionalität hinzu

7. Interaktion

- Dekorierer leitet Anfragen an Komponentenobjekt weiter
- kann optional vor oder nach dem Weiterleiten zusätzliche Operationen ausführen

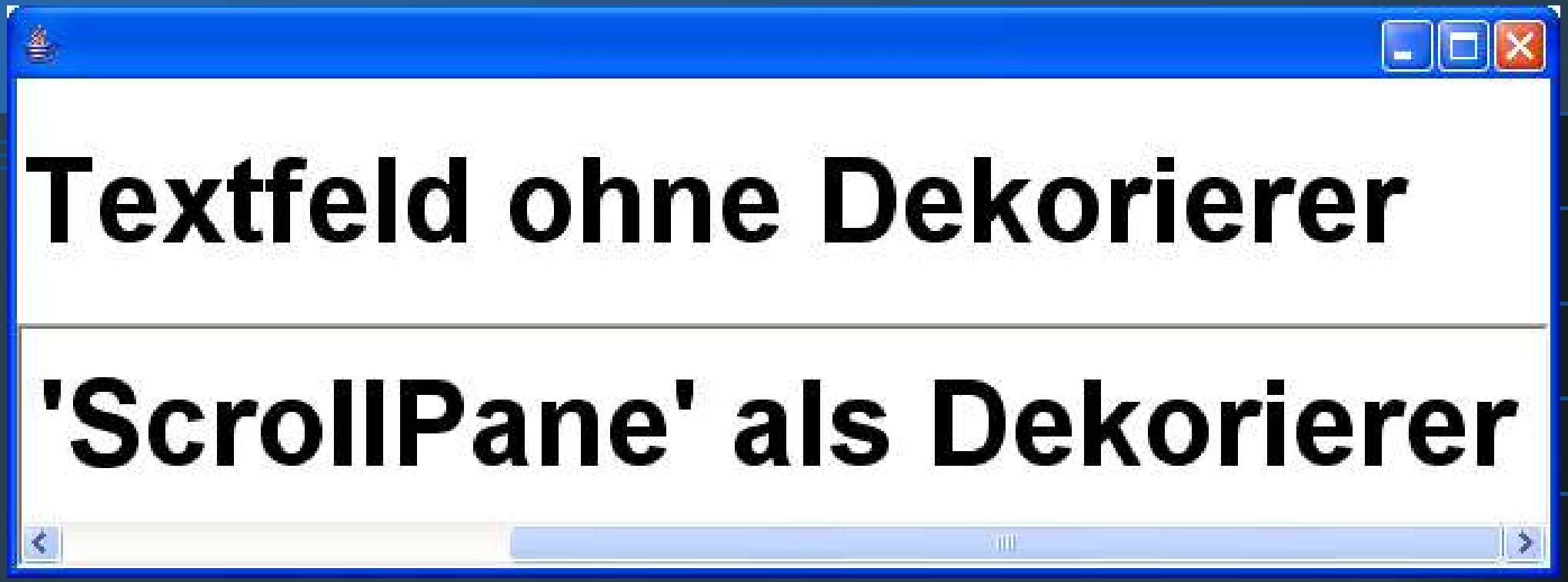
8. Konsequenzen

- größere Flexibilität im Vergleich zu statischer Vererbung
- vermeidet es, in der Hierarchie oben stehende Klassen mit Funktionalität zu überfrachten
- ein Dekorierer und seine Komponenten sind nicht identisch
- viele kleine Objekte, d.h. Komplikationen beim Verstehen sowie Debuggen

9. Implementierung

- Schnittstellenkonformanz
- Weglassen der abstrakten Dekoriererklasse
- Komponentenklasse leichtgewichtig lassen
- Abwägen von Vor- und Nachteilen

10. Beispiel I



10. Beispiel II

```
private void initComponents() {
    ...
    /* fügt das Label dem Frame hinzu */
    add(label1);
    ...
    /* fügt das Label der ScrollPane hinzu
     * ScrollPane hat dieselbe Schnittstelle
     * wie Frame */
    scrollPane1.add(label2);
    add(scrollPane1);
    ...
 }
```

11. Bekannte Verwendungen

- hauptsächlich bei GUI
- Komprimierung von Streamdaten
- SSH-Tunneling

12. Verwandte Muster

- Adapter
- Kompositum
- Strategie

13. Tschüss

Danke für die Aufmerksamkeit
bis zum nächsten Mal