

Kategorien von Entwurfsmustern

- Muster für das Erzeugen von Instanzen
 - Singleton
- strukturelle Muster zur Komposition von Klassen oder Objekten
 - Composite
 - Adapter
 - Decorator
- Verhaltensmuster betreffen Interaktion von Klassen oder Objekten
 - Command

Bestandteile eines Entwurfsmusters

- **Name** (kurz und beschreibend)
- **Problem**: *Was das das Entwurfsmuster löst*
- **Lösung**: *Wie es das Problem löst*
- **Konsequenzen**: Folgen und Kompromisse des Musters.

- es soll nur eine einzige Instanz einer Klasse geben, die global verfügbar sein soll
- Beispiele:
 - zentrales Protokoll-Objekt, das Ausgaben in eine Datei schreibt.
 - Druckaufträge, die zu einem Drucker gesendet werden, sollten nur in einen einzigen Puffer geschrieben werden.

Entwurfsmuster Singleton (Einzelstück)

Lösung (Muster für das Erzeugen von Instanzen):

Singleton
<ul style="list-style-type: none">- static instance: Singleton- static ...
<ul style="list-style-type: none">+ static Singleton getInstance()- Singleton()+ ...

Code

```
1 public final class Singleton {
2
3     private static Singleton instance;
4     // speichert einzige Instanz
5
6     private Singleton() {}
7     // kann von außerhalb nicht benutzt werden
8
9     // liefert einzige Instanz
10    public synchronized static Singleton getInstance() {
11        if (instance == null) { // lazy instantiation
12            instance = new Singleton();
13        }
14        return instance;
15    }
16 }
```

- Singleton hat strikte Kontrolle über wie und wann Klienten es verwenden
- vermeidet globale Variablen
- leicht erweiterbar, um mehrere Instanzen zuzulassen
- Singleton kann spezialisiert werden

Spezialisierung von Singletons

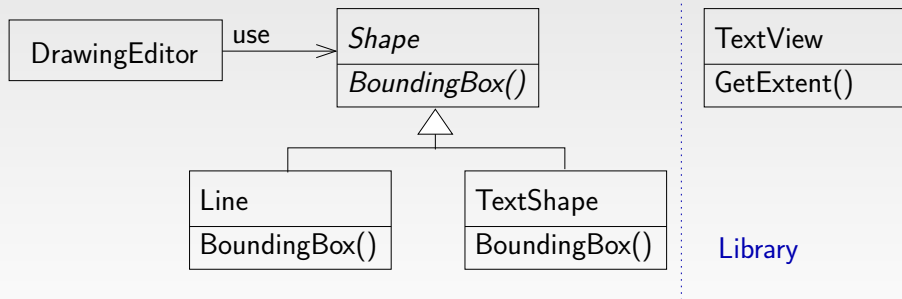
```
1 public class Singleton {
2     protected Singleton() {}
3     // kann von außerhalb nicht benutzt werden
4     private static Hashtable<String, Singleton> registry
5         = new Hashtable<String, Singleton>();
6     // Registry für alle Instanzen von Singleton und Unterklassen
7
8     protected static String ClassKey () {return "Singleton";};
9     // eindeutiger Schlüssel der Klasse;
10    // muss von Unterklasse redefiniert werden
11
12    // liefert einzige Instanz
13    public synchronized static Singleton getInstance()
14        {return registry.get (ClassKey ());};
15
16    // muss vor Benutzung aufgerufen werden;
17    // registriert Instanz; muss von Unterklasse redefiniert werden
18    public static void Init ()
19        {registry.put (ClassKey (), new Singleton());};
20 }
```

Spezialisierung von Singletons

```
1 public class DerivedSingleton extends Singleton {  
2  
3     protected DerivedSingleton() {};  
4  
5     protected static String ClassKey ()  
6         {return "DerivedSingleton";};  
7  
8     public static void Init ()  
9         {registry.put (ClassKey (), new DerivedSingleton());};  
10  
11 }
```

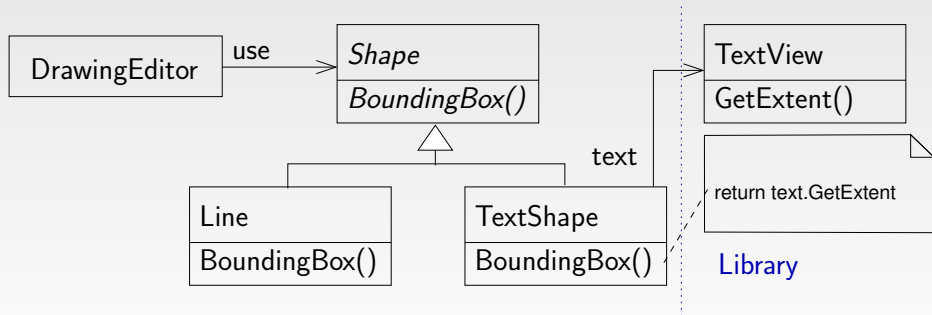

Problem

- eine vorhandene wiederverwendbare Komponente hat nicht die passende Schnittstelle,
- und der Code der Komponente kann nicht verändert werden

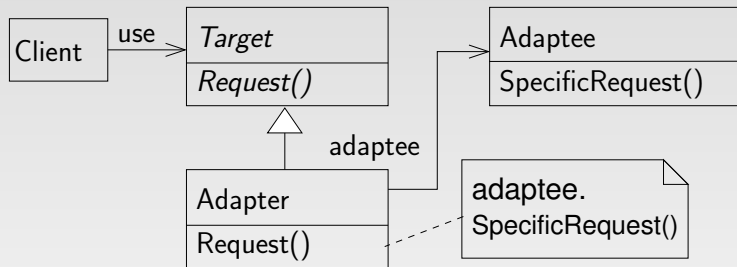


Problem

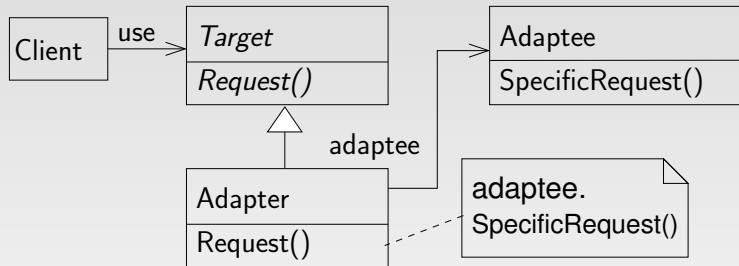
- eine vorhandene wiederverwendbare Komponente hat nicht die passende Schnittstelle,
- und der Code der Komponente kann nicht verändert werden



Lösung: Objekt-Adapter



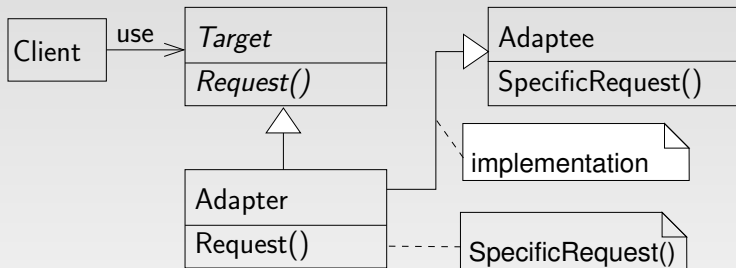
Lösung: Objekt-Adapter



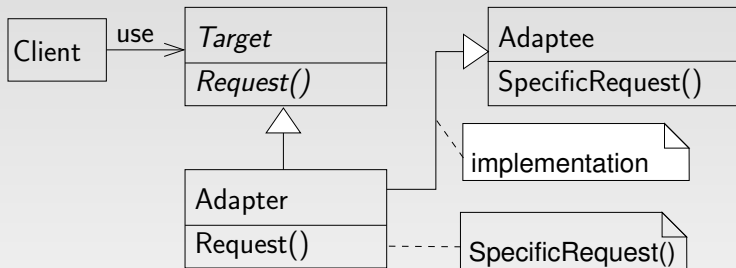
Konsequenzen:

- erlaubt Anpassung von Adaptee und all seine Unterklassen auf einmal
- Overriding von Adaptees Methoden schwierig
 - Ableitung von Adapter
 - Adapter referenziert auf Ableitung
- führt Indirektion ein

Lösung: Klassen-Adapter



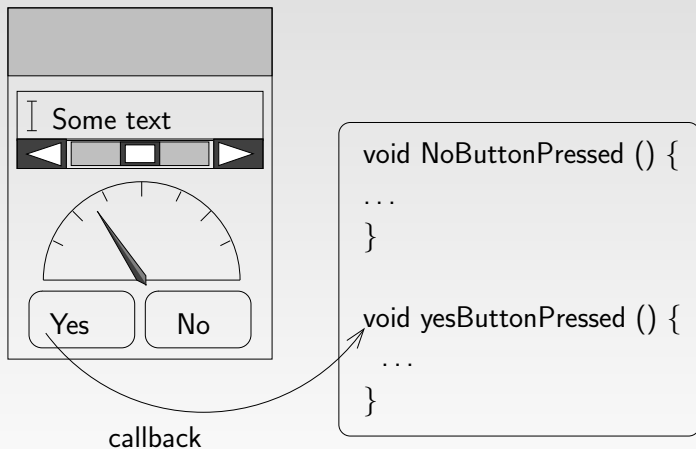
Lösung: Klassen-Adapter



Konsequenzen:

- keine Indirektion
- setzt Mehrfachvererbung voraus
- passt nur Adaptee an, nicht seine Unterklassen
- Overriding von Adaptees Methoden einfach

Problem



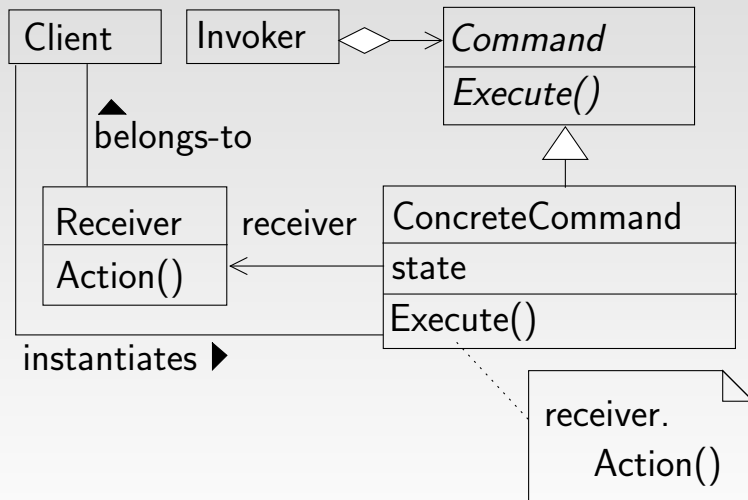
Lösung in C: Klient

```
1 void YesButtonPressed () {...}
2 void NoButtonPressed () {...}
3
4 Button mybutton;
5
6 int main (){
7     attach (&mybutton, &YesButtonPressed);
8     ...
9     event_loop ();
10 }
```

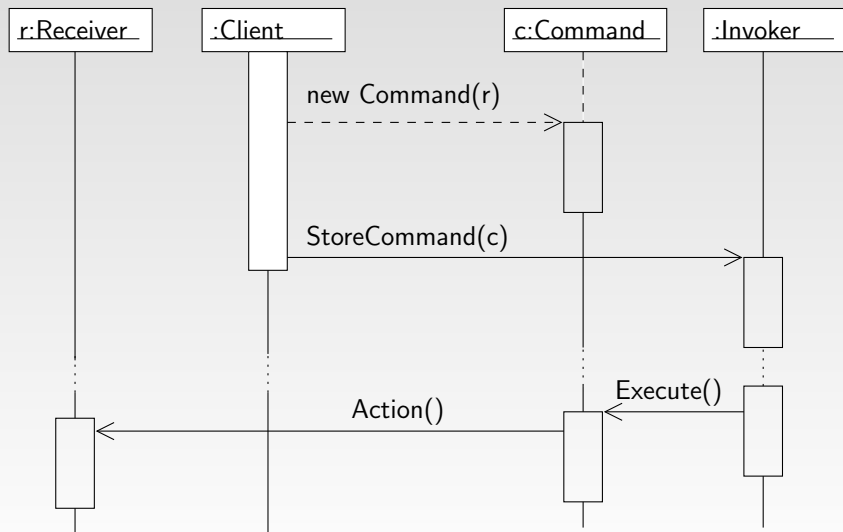

Lösung in C: Mechanismus

```
1 typedef void (*Callback)();
2
3 typedef struct Button {
4     Callback execute;
5     int x;
6     int y;
7 } Button;
8
9 void attach (Button *b, Callback execute) {
10     b->execute = execute;
11 }
12
13 void event_loop () {
14     ...
15     widgets[i]->execute();
16     ...
17 }
```

Lösung mit OOP



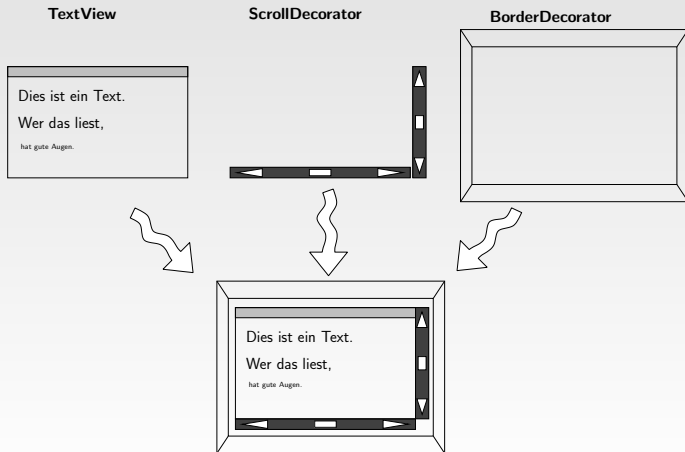
Lösung mit OOP



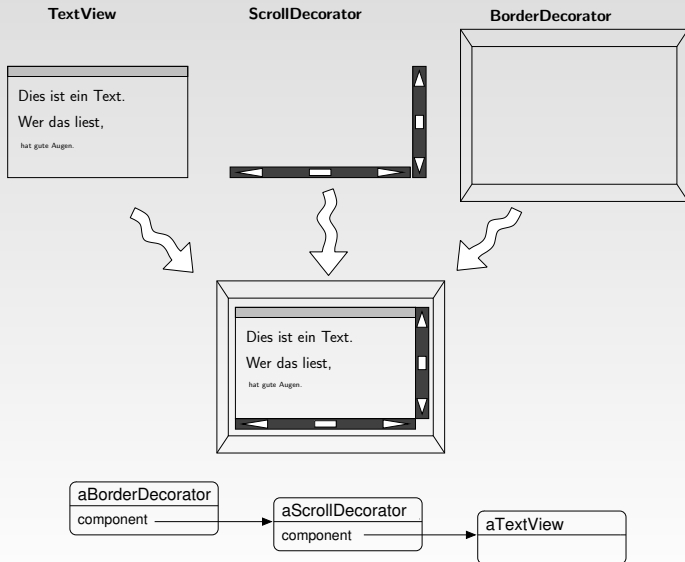
- Entkopplung von Objekt, das Operation aufruft, von dem, welches weiß, wie man es ausführt
- Kommandos sind selbst Objekte und können als solche verwendet werden (Attribute, Vererbung etc.)
- Hinzufügen weiterer Kommandos ist einfach

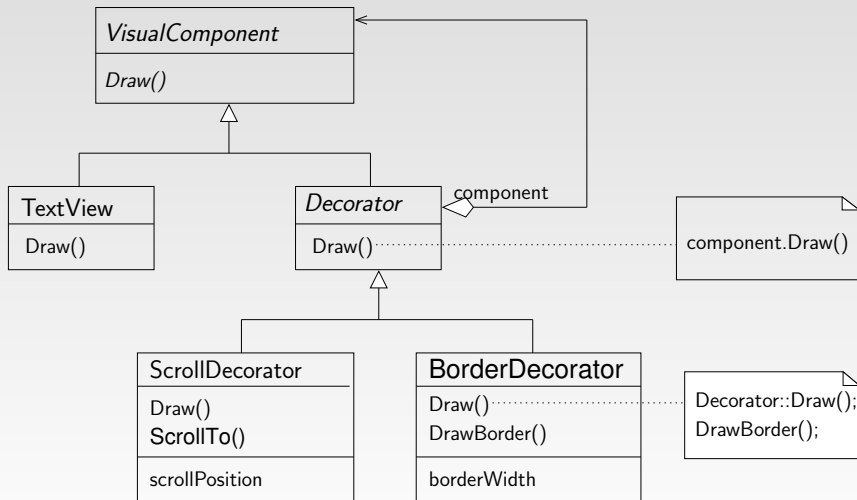
Problem

- Eigenschaften sollen einzelnen Objekten, nicht ganzen Klassen, zugewiesen werden können
- Zuweisung soll dynamisch geschehen



Problem





- höhere Flexibilität als statische Vererbung
- vermeidet Eier-Legende-Wollmilchsau-Klassen
- Dekorator und dekoriertes Objekt sind nicht identisch
- vielfältige dynamische Kombinationsmöglichkeiten → schwer statisch zu analysieren

2 Software-Produktlinien

- Lernziele
- Software-Wiederverwendung
- Erfolgsgeschichten
- Definition
- Übersicht
- Kostenaspekte
- Practice Areas
- Entwicklung der Core-Assets
- Produktentwicklung
- Essentielle Aktivitäten
- Einführung von Produktlinien
- Implementierungsstrategien
- Schwierigkeiten
- Wiederholungsfragen

Software-Produktlinien

- Definition und Bedeutung
- Vor- und Nachteile
- Technische Aspekte
- Organisatorische Aspekte

N.B.: Basiert auf Folien von Linda Northrop

<http://www.sei.cmu.edu/productlines/presentations.html>

1960: Unterprogramme

1970: Module

1980: Objekte

1990: Komponenten

→ opportunistische Wiederverwendung im Kleinen; hat nicht den erwarteten Erfolg gebracht

1960: Unterprogramme

1970: Module

1980: Objekte

1990: Komponenten

→ opportunistische Wiederverwendung im Kleinen; hat nicht den erwarteten Erfolg gebracht

Software-Produktlinien: geplante Wiederverwendung auf allen Ebene für Familien ähnlicher Systeme

- Komponenten
- Software-Dokumentation
- Architektur
- Tests (unter anderem Integrations-, Leistungs- und Komponententests)
- Anforderungsspezifikation
- Entwicklungsprozess, Methoden und Werkzeuge
- Budget-/Zeit- und Arbeitspläne
- Handbücher
- Entwickler