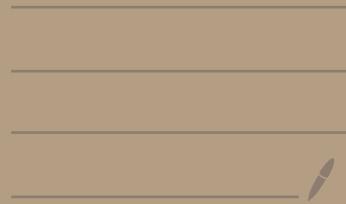


# Entwurfsmuster

---



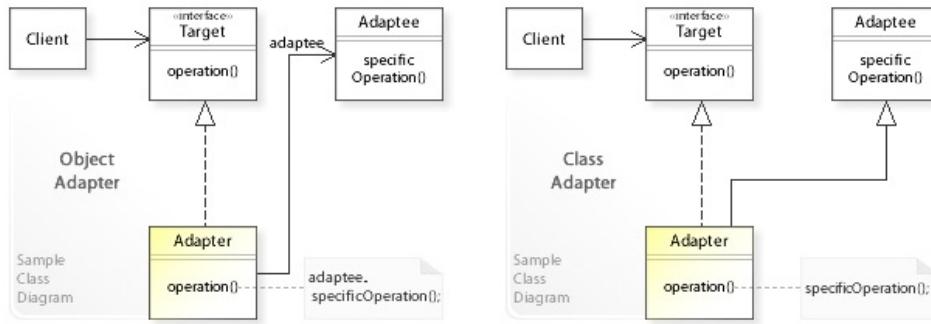
# Entkopplungsmuster

- teilen System in mehrere Einheiten
- Einheiten können unabh. erstellt, ausgetauscht, wieder verwendet werden
  - ↳ lokale Änderungen möglich ohne ganzes System zu modifizieren

## Adapter

- passe Schnittstelle einer Klasse an erwartetes Verhalten einer anderen an
  - ↳ Klassen arbeiten zusammen die eig. inkompatibel

## Objectadapter



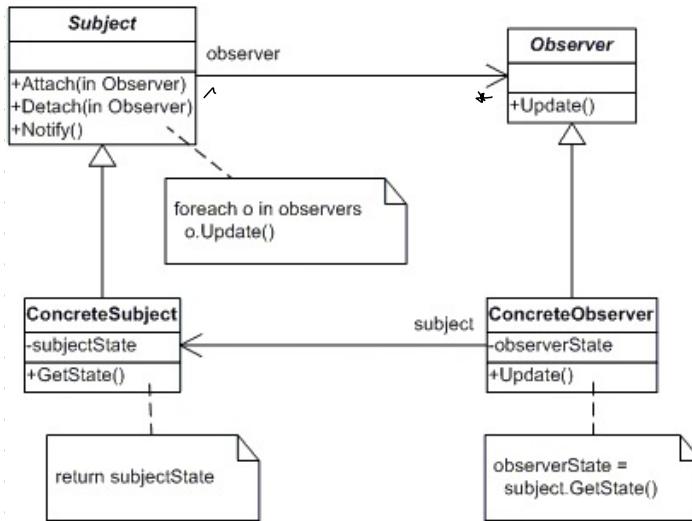
## Anwendung

- ex. Klassen sollen verwendet werden, aber (unveränderl.) Schnittstelle nicht kompatibel
- Klassen werden wieder verwendbar unabh. von Schnittstelle

Objectadapter: verwende ex. Unterklassen ohne alle Schnittstellen anzupassen

## Observer

- 1-zu-n Abh. zw. Objekten → Änderung eines Objekts führt zu Benachrichtigung + autom. Aktualisierung aller abh. Objekte

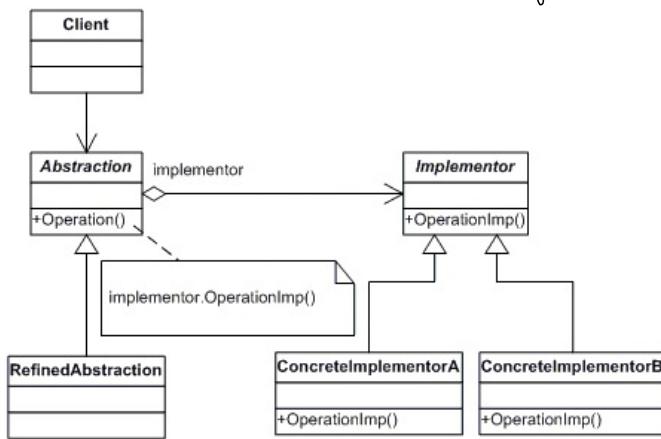


## Anwendung

- Änderung eines Objekts verlangt Änderung anderer, aber unbekannt welche wie viele
- Objekt muss andere benachrichtigen ohne Annahmen über diese zu treffen

## Bridge

Entkoppelt Abstraktion von ihrer Implementierung

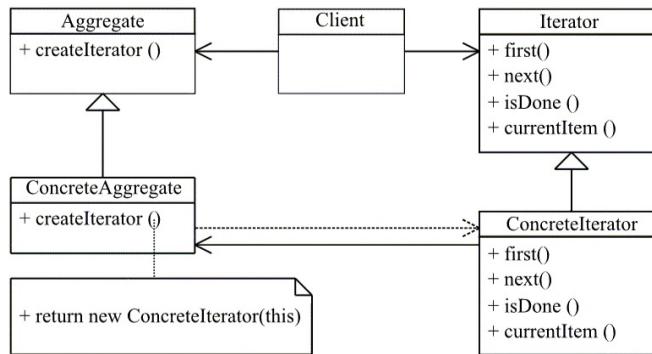


## Anwendung

- Vermeidung dauerhafter Verbindung zw. Abstraktion, Implementierung
- Impl. einer Abstraktion soll vollst. vom Client verdeckt werden
- Impl. soll von mehreren Objekten genutzt werden

## Iterator

- Sequentieller Zugriff auf Elemente eines zusammengesetzten Objekts ohne zugrundeliegende Repräsentation offen zu legen

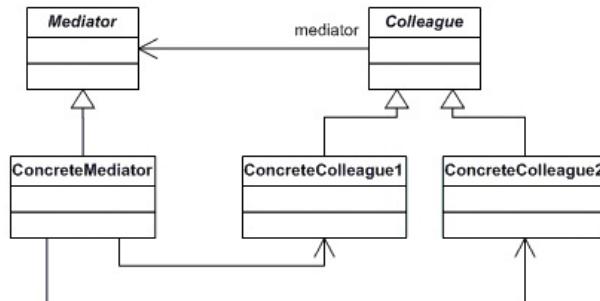


## Anwendung

- Zugriff auf zusammengesetztes Objekt ermöglichen ohne interne Struktur offen zu legen
- einheitliche Schnittstelle zur Traversierung zusammengesetzter Objekte

## Mediator

- Objekt kapselt Zusammenspiel einer Menge von Objekten
  - fordert lose Kopplung: Objekte nehmen nicht explizit Bezug aufeinander

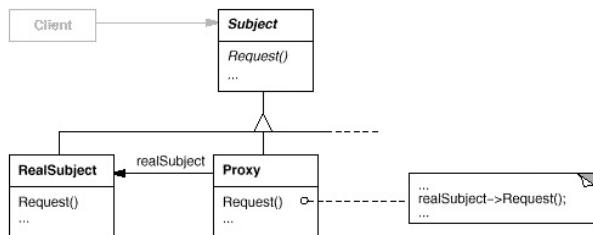


## Anwendung

- wenn viele Objekt in wohldef. aber komplexer Weise zusammenarbeiten
- Verhalten maßschneidern das auf mehrere Klassen verteilt ohne viele Unterklassen

## Proxy

kontrolliere Objektzugriff mit vorgelagertem Stellvertreterobjekt



## Anwendung

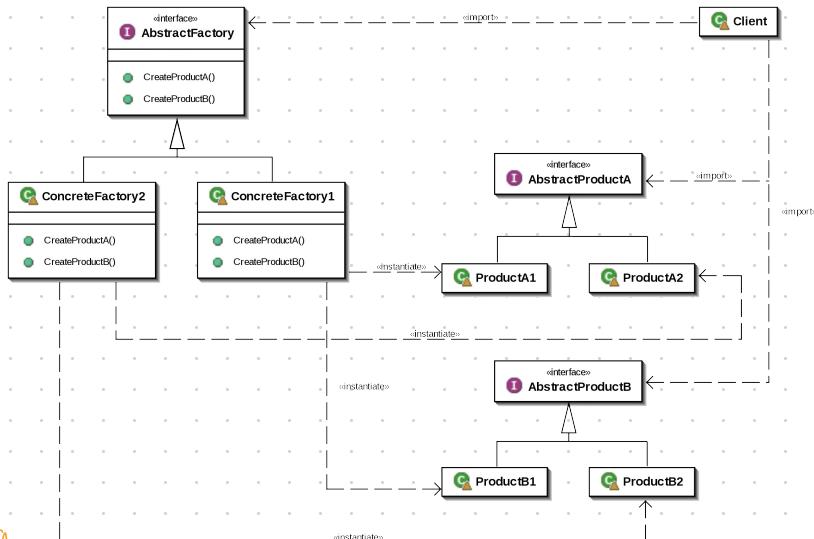
- protokolliernder Proxy: zählt Referenzen auf eigenes Objekt
- Caching Proxy: lädt persistentes Objekt erst in Speicher wenn erster Mal angesprochen
- remote Proxy: lokaler Stellvert. für Objekt in anderem Adr. Raum
- Virtual Proxy: erzeugt teure Objekte auf Verlangen
- Firewall: Kontrolle + Zugriff auf Orig. Obj.
- synchr. Proxy: koord. Zugriff mehrerer Threads auf ein Obj.

## Varianten-Muster

Gemeinsamkeiten aus verwandten Einheiten rausziehen, an einzelner Stelle beschreiben → verweicht Redundanzen

## Abstract factory

Schnittst. zum Erzeugen von Familien verwandter, voneinander unabh. Objekte ohne ihre konkreten Klassen zu benennen

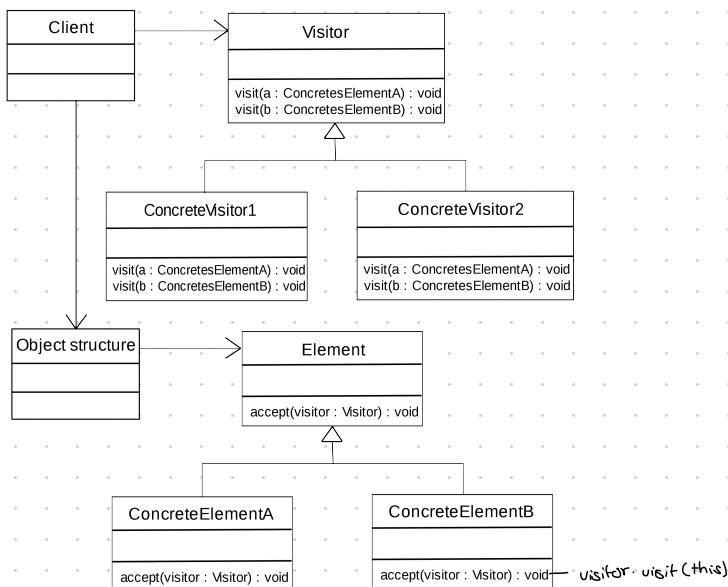


## Anwendung

- System soll unabh. von Produktzerzeugung - Zusammensetzung, - repr. sein  
 System soll mit einer von mehreren Produktfam. konfig. werden

## Visitor

Auf Element einer Objektstruktur auszuführende Op. als Objekte kapseln

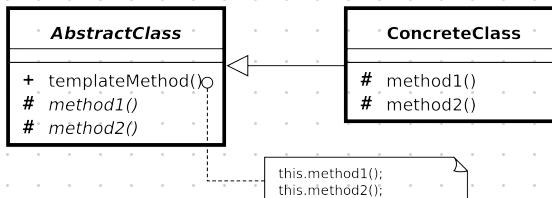


## Anwendung

- Objektstruktur enth. viele Klassen von Obj. mit untersch. Schnittstellen, auszuführende Op. hängt von konkreten Klassen ab
- mehrere untersch. nicht-verwandte Op. auf Obj. einer Datenstruktur ausführen ohne diese "zu verschmutzen"
- Klassen einer Obj. struktur ändern sich gelten, aber oft neue Operationen

## Template Method

definiere Skellett eines Algos in einer op., delegiere einzelne Schritte an Unterklassen → Unterkl. können bestimmte Schritte ändern / überschreiben ohne Struktur zu verändern

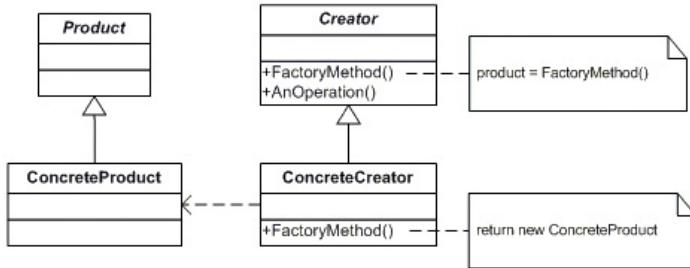


## Anwendung

- Invarianten Teil des Algos einmal def., Unterklassen impl. variierendes Verhalten ↳ gem. Verhalten austauschen → vermeidet Redundanz
- erlaube Erweiterungen nur an best. Stellen

## Factory method

def. Klassenschnittstelle mit Op. zur Objekterzeugung → Unterklasse entscheidet von welcher Klasse erstzeugtes Obj. ist



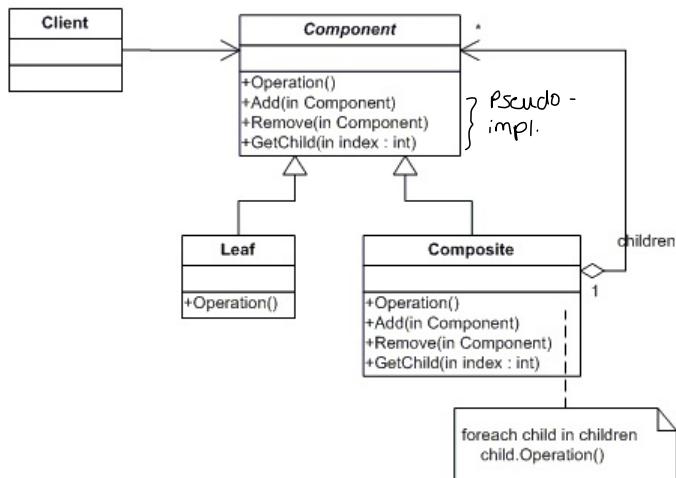
## Anwendung

- eine Klasse kennt Klasse von Obj. die sie erzeugen muss nicht im Voraus
- Klasse soll Zuständigkeiten an Hilfsunterklassen delegieren
- ↪ Wissen an welche soll lokalisiert werden

## Composite

füge Obj. zu Baumstrukturen zum um Bestandshierarchien zu repräsentieren  
 ↪ Client kann Obj. / Aggregat einheitlich behandeln

Composite isoliert gemeinsame Eigenschaften, bildet daraus Oberklasse /

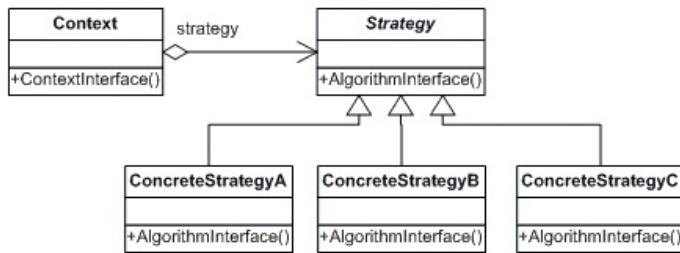


## Anwendung

- Repr. von Bestandshierarchien von Objekten
- Client soll in der Lage sein untersch. zw. zusammgesetzten, einzelnen Objekten zu ignorieren

## Strategy

Definiere Familie von Algos, kapsle sie und mache sie austauschbar  
 ↳ können unabh. von nutzenden Clients (kontext) variiert werden

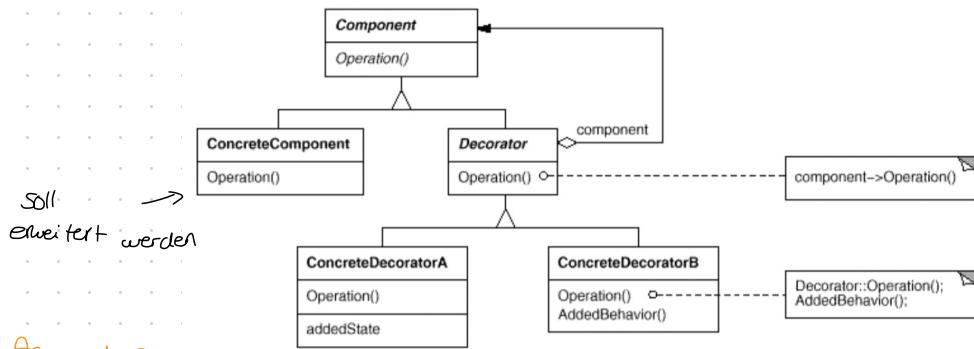


## Anwendung

- viele verwandte Klassen unterscheiden sich nur im Verhalten
- versch. Varianten eines Algos benötigt
- Algo verwendet Datenstruktur die Client nicht bekannt sein soll
- Klasse def. untersch. Verhaltensweisen → Strategie statt Falluntersch.

## Decorator

Fügt dynamisch neue Funkt. zu Objekten hinzu  
 ↳ Alternative zur Vererbung



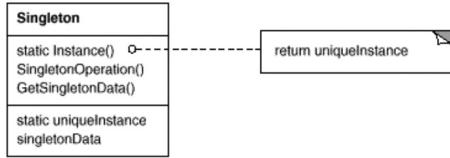
## Anwendung

- hinzufügen neuer Funktionalität ohne Subjekt zu verändern
- z.B. Schnellstelle erweiter

# Zustandspattern

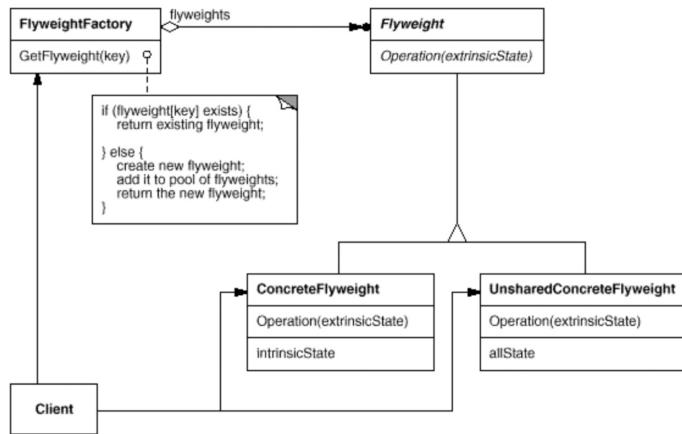
## Singleton

- Klasse besitzt genau eine Instanz mit globalem Zugriffspunkt



## Flyweight

- Objekte kleinster Granularität gemeinsam nutzen um große Menge davon effizient speichern zu können

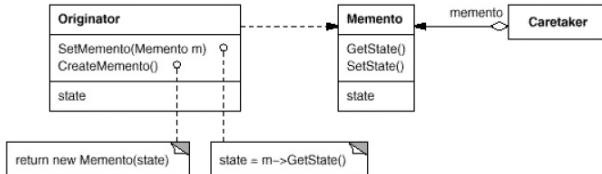


## Anwendung

- Anwendung verwendet große Menge von Obj. → hohe Speicherkosten

## Memento

- Erfassen, externalisieren des Objektzustands ohne Kapselung zu verletzen

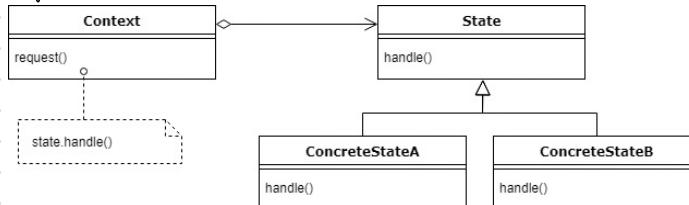


## Anwendung

- aktueller Obj. Zustand muss gespeichert werden um Objekt später zurückzusetzen  
wenn direkte Schnittstelle zum Zustand impl. details offenlegen würde

## State

Ändert Obj. verhalten wenn sich interner Zustand ändert

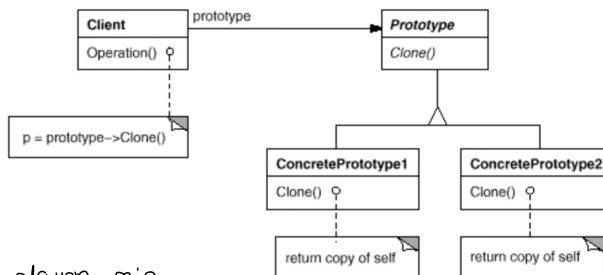


## Anwendung

Objektverhalten abh. vom aktuellen Zustand

## Prototype

Bestimme Arten der zu erzeugenden Objekte durch Verwendung eines typischen Exemplars  
 ↳ neue Objekte durch Kopieren dieses Prototyps



## Anwendung

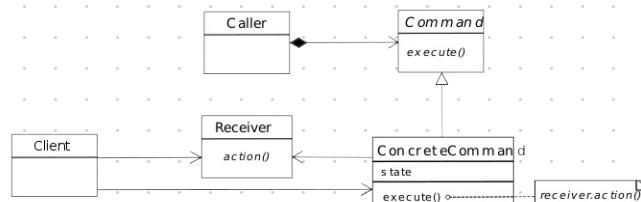
- System soll unab. davon sein wie Produkte erstellt, repr., zugesetzt werden
- falls Aufbau eines Objekts deutl. aufwändiger als Kopie
- falls Klassen zu erstehendes Obj. erst zur Laufzeit spezifiziert

## Steuerungsmuster

Steuern Kontrollfluss, bewirken dass zur richtigen Zeit richtige Methoden aufgerufen

## Command

Befehl als Objekt kapseln → Client mit versch. Anfragen parametrisieren, Operationen in Queue, führe Logbuch, mache Op. rückgängig

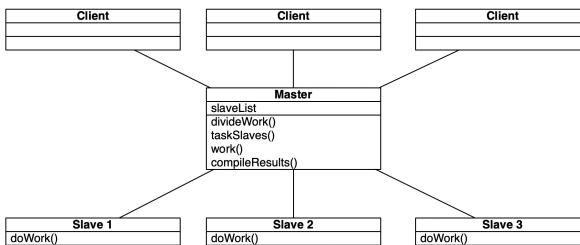


## Anwendung

- undo einer Op. soll unterstützt werden
- Protokollieren der Änderungen soll unterstützt werden
- Anfragen zu untersch. Zeitpunkten spezifizieren, aufrufen, ausführen

## Master / Slave

- fehler tolerant, parallele Berechnung
- Master verteilt Aufgaben an identische Slaves
- Master berechnet Ergebnis aus Teilergebnissen der Slaves



## Anwendung

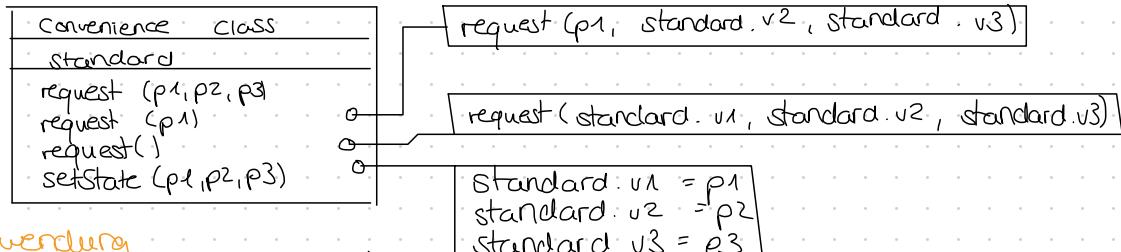
- mehrere Aufgaben können unabh. bearbeitet werden  
↳ zB mit mehreren parallelen Prozessoren

## Bequemlichkeitsmuster

sparen Schreib- / Denkarbeit

### Convenience Class

vereinfache Methodenauftrag durch Bereithalten der Params in Spezieller Klasse



## Anwendung

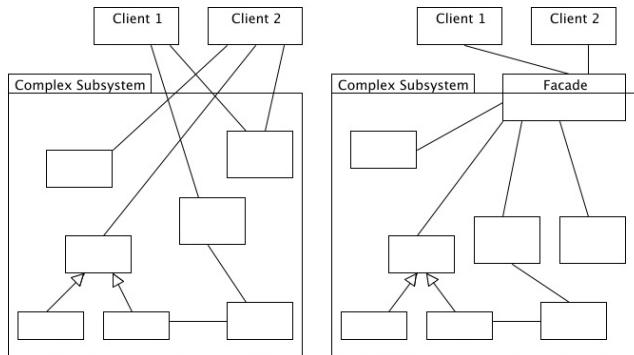
- Methoden häufig mit gleichen Params aufgerufen die sich selten ändern

### Convenience method

vereinfache Methodenauftrag durch Bereitstellung der Params in Methoden

## Facade

einheitl. Schnittstelle zu Menge von Schnittst. eines Subsystems

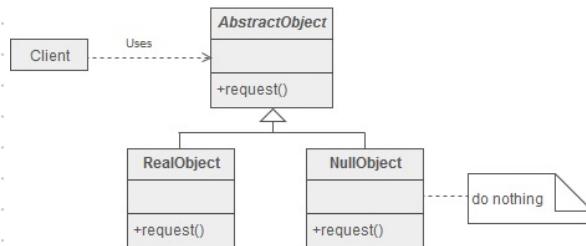


## Anwendung

- einfache Schn. zu komplexen Subsystemen
- viele Abh. zw. Clients und Impl. Klassen einer Abstraktions Schichten → Fassade = Eingangspunkt
- Einteilung der Subsys. in

## Null object

Stellvertreter mit gleicher Schn. stelle der nichts tut  
↳ kapselt Impl.-Entscheidungen



## Anwendung

- Objekt benötigt Mitarbeiter, einer / mehrere sollen nichts tun
- Client soll untersch. zw. echtem Objekt und Nullobjekt nicht kennen