Zusammenfassung: Entwurfsmuster

1. Definition Entwurfsmuster

- Def: Ein Software-Entwurfsmuster beschreibt eine Familie von Lösungen für ein Software-Entwurfsproblem.
- Wiederverwendung von Entwurfswissen

2. Wozu Entwurfsmuster?

- Verbessern die Kommunikation im Team (einheitliche Terminologie)
- Bringe komplexe Konzepte in verständliche Form, Dokumentieren und verdeutlichen Entwurfswissen
- „Vermeiden die Neuerfindung des Rads“
- Verbessernd Code-Qualität und Code-Struktur

3. Kategorien von Entwurfsmustern

3.1 Entkopplungsmuster

- Entkopplungs-Muster teilen ein System in mehrere Einheiten, so dass einzelne Einheiten unabhängig voneinander erstellt, verändert, ausgetauscht und wiederverwendet werden können
- Vorteil: lokale Änderungen/Anpassungen möglich
- Kopplungsglied lässt entkoppelte Einheiten über Schnittstelle modifizieren

3.1.1 Adapter

♥ Zweck:
- Passe Schnittstelle an von dem Klienten erwartete Schnittstelle an
- Lässt Klassen zusammenarbeiten, die sonst nicht dazu in der Lage wären

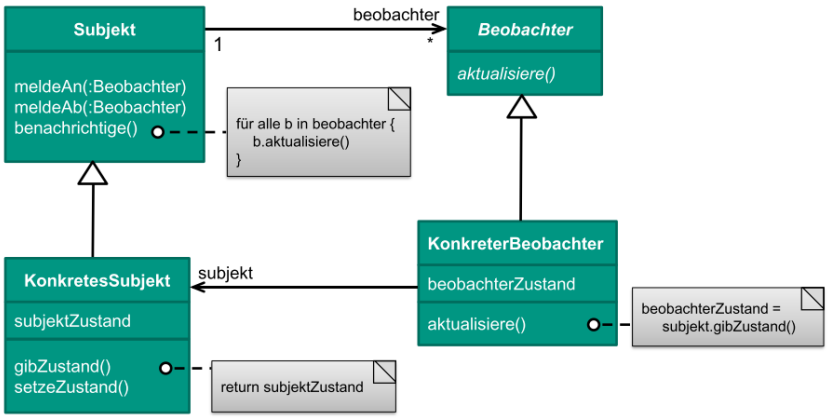
♥ Struktur:
- Ohne Mehrfachvererbung (Objektadapter): Adapter erbt von Zielschnittstelle, ruft passende Operation des adaptierten Objekts auf
- Mit Mehrfachvererbung (Klassenadapter): Adapter erbt von Zielschnittstelle und Adaptierter Klasse, ruft spezifische Operation der Adaptierten Klasse für Zieloperation auf

♥ Anwendbarkeit:
- ex. Klasse soll verwendet werden, Schnittstelle unpassend und kann nicht geändert werden
- wiederverwendbare Klasse soll erstellt werden, soll mit nicht vorhersehbaren Klassen zusammenarbeiten
- verschiedene ex Unterklassen sollen verwendet werden, aber nicht alle Schnittstellen der Unterklassen sollen angepasst werden

3.1.2 Beobachter (engl. observer)

♥ Zweck:
- Definiert 1-zu-n Abhängigkeit zwischen Objekten
- Änderung eines Zustandes eines Objekts führt zu Benachrichtigung aller abhängigen Objekte

♥ Struktur:



♥ Anwendbarkeit:
- Wenn Änderung eines Objekts die Änderung einer unbekannten Menge anderer Objekte erfordert
- Wenn Objekt andere Objekte benachrichtigen muss

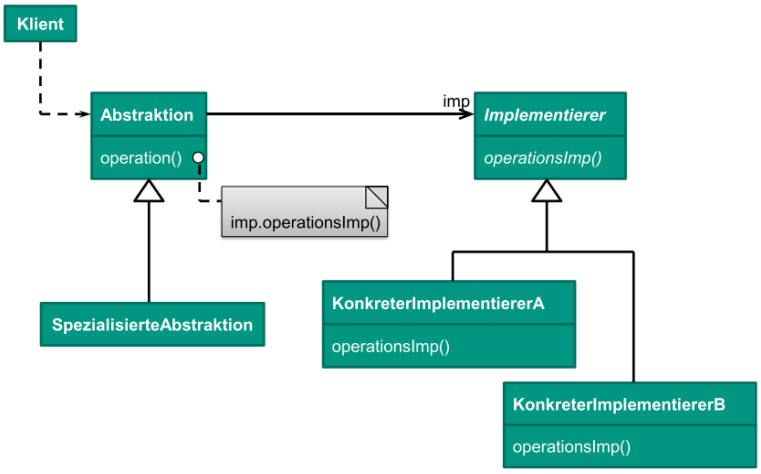
♥ Konsequenzen:
- Subjekte und Beobachter können unabhängig voneinander wiederverwendet werden
- Beobachter können geändert werden ohne das Subjekt/ andere Beobachter zu ändern
- Subjekt und Beobachter gehören versch. Schichten der Benutzt-Hierarchie an, erzeugen aber keine Zyklen
- Automatischer Rundruf von Änderungen
- Beobachter entscheiden, ob Benachrichtigung ignoriert wird

♥ Implementierung:
- mehr als ein Subjekt: aktualisiere(s: Subjekt)
- Auslösung der Aktualisierung: setzeZustand() ruft benachrichtige() auf
- Löschen eines Beobachters: Zuerst beim Subjekt abmelden
- Aktualisierung: Pull- und Push-Modell
  Pull-Modell: Beobachter holt Daten vom Subjekt (evtl. ineffizient)
  Push-Modell: Subjekt schickt Daten an Beobachter (verringert evtl. Wiederverwendbarkeit)

3.1.3 Brücke (engl. bridge)

♥ Zweck:
- Entkopple Abstraktion von ihrer Implementierung
- Beide können unabhängig voneinander variiert werden

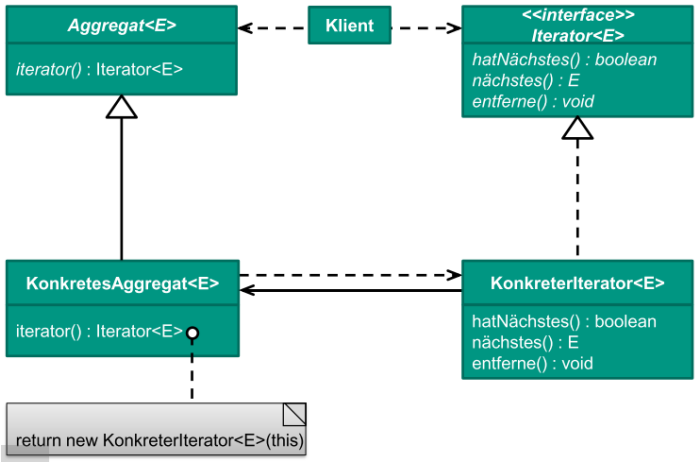
♥ Struktur:



- ♥ Anwendbarkeit:
  - Wenn dauerhafte Verbindung zwischen Abstraktion und Implementierung vermieden werden soll
  - Sowohl von Abstraktion als auch von Implementierungen sollen Unterklassen gebildet werden können
  - Änderungen in der Implementierung einer Abstraktion sollen keine Auswirkung auf Klienten haben
  - Implementierung einer Abstraktion soll vom Klienten versteckt werden
  - Implementierung soll von mehreren Objekten aus gemeinsam benutzt werden

3.1.4 Iterator

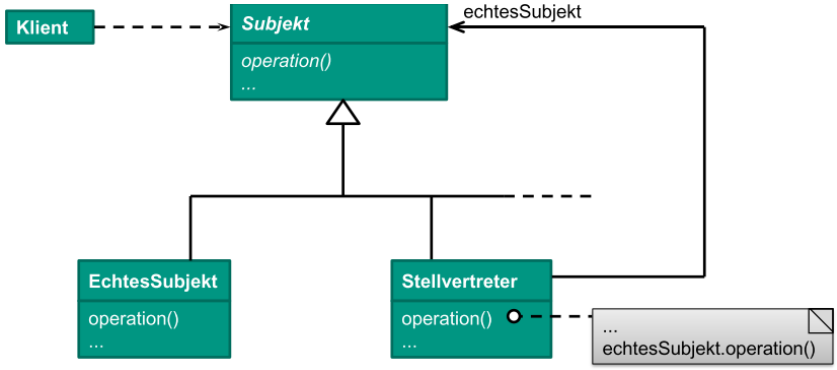
- ♥ Zweck:
  - Ermöglicht sequentiellen Zugriff auf Elemente eines zusammengesetzten Objekts
  - Ohne zugrundeliegende Repräsentation des Objekts offenzulegen
- ♥ Struktur:



- ♥ Anwendbarkeit:
  - Erlaubt Zugriff auf Inhalt ohne Struktur des Objekts offenzulegen
  - Um einheitliche Schnittstelle für versch. Zusammengesetzte Strukturen anzubieten
  - Robuster Iterator: Erlaubt gleichzeitig mehrere Traversierungen

3.1.5 Stellvertreter

- ♥ Zweck:
  - Kontrolliert Zugriff auf ein Objekt mithilfe eines vorgelagerten Stellvertreterobjekts
- ♥ Struktur:

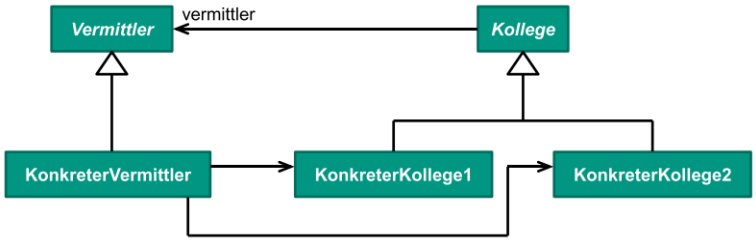


- ♥ Anwendbarkeit:
  - Bei Bedarf nach anpassungsfähigeren/intelligenteren Referenz als einfachen Zeiger
  - Protokollierender Stellvertreter: Zählt Referenzen auf das Objekt (Objekt kann freigegeben werden sobald keine Referenzen darauf existieren)

- Puffernder Stellvertreter: Lädt Objekt erst dann in Speicher, wenn es das erste Mal angesprochen wird
- Fernzugriffsvertreter: Stellt lokalen Stellvertreter für Objekt in anderem Adressraum dar
- Platzhalter: Erzeugt teure Objekte auf Verlangen
- Firewall: Kontrolliert Zugriff auf Originalobjekt (versch. Zugriffsrechte möglich)
- Synchronisierungsvertreter: Koordiniert Zugriff mehrerer Fäden auf ein Objekt
- Dekorierer: Fügt zusätzliche Zuständigkeiten zu bestehendem Objekt hinzu

3.1.6 Vermittler

- ♥ Zweck:
  - Definiert ein Objekt, welches Zusammenspiel einer Menge von Objekten in sich kapselt
  - Vermittler fördern lose Kopplung (Objekte nehmen nicht direkt aufeinander Bezug)
- ♥ Struktur:



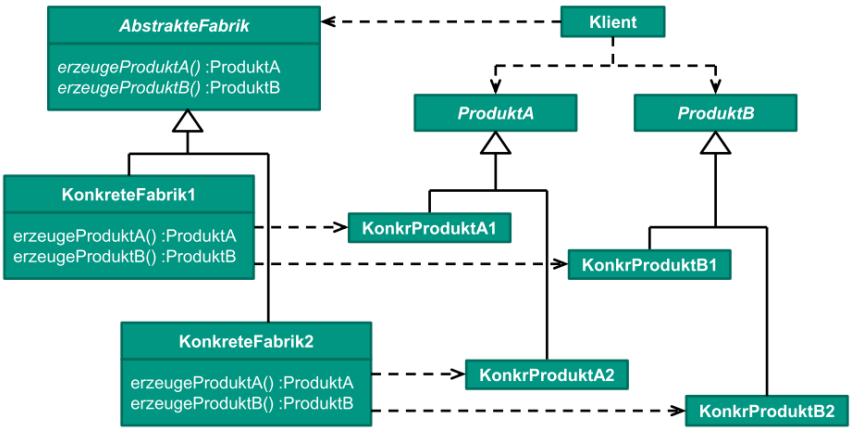
- ♥ Anwendbarkeit:
  - Wenn Menge von Objekten vorliegt, die in komplexer Weise zusammen arbeiten (unstrukturierte, komplexe Abhängigkeiten)
  - Wenn Wiederverwertung eines Objekts schwierig, weil eng mit anderen verbunden
  - Auf mehrere Klassen verteiltes Verhalten soll maßgeschneidert werden, ohne Unterklassen zu bilden

3.2 Varianten-Muster

- Gemeinsamkeiten von verwandten Einheiten werden herausgezogen und an einer einzigen Stelle beschrieben
- Vorteil: Unterschiedliche Komponenten können dann einheitlich verwendet werden, Code-Wiederholungen werden vermieden

3.2.1 Abstrakte Fabrik (engl. abstract factory)

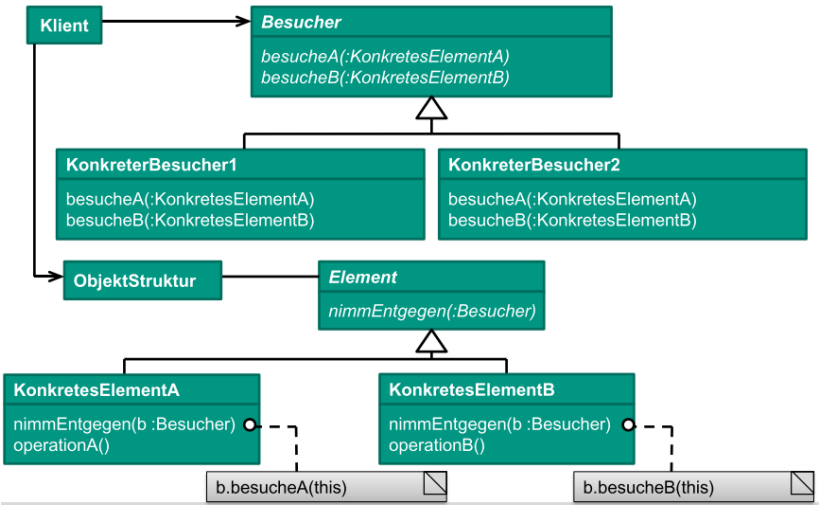
- ♥ Zweck:
  - Bietet Schnittstelle zum Erzeugen von Familien verwandter Objekte, ohne konkrete Klassen zu benennen
- ♥ Struktur:



- ♥ Anwendbarkeit:
  - System soll unabhängig davon sein, wie seine Produkte erzeugt/ zusammengesetzt / repräsentiert sind
  - Wenn System mit einer von mehreren Produktfamilien konfiguriert werden soll
  - Wenn Familien von aufeinander abgestimmten Produktobjekten zusammen verwendet werden soll
  - Bei Klassenbibliothek, die nur Schnittstellen (keine Implementierungen) offenlegt

3.2.2 Besucher

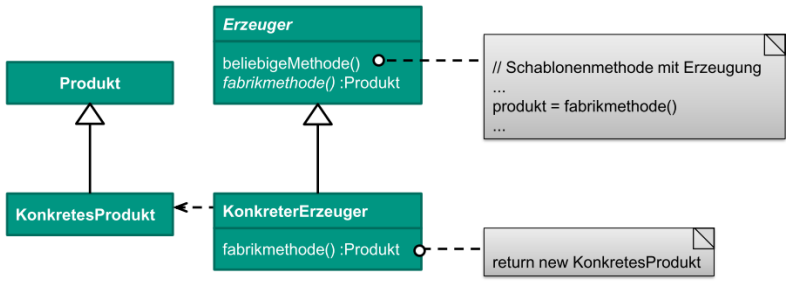
- ♥ Zweck:
  - Kapsle eine auf den Elementen einer Objektstruktur auszuführende Operation als ein Objekt
  - Ermöglicht Definition einer neuen Operation, ohne Klassen der von ihr bearbeiteten Elemente zu verändern
- ♥ Struktur:



- ♥ Anwendbarkeit:
  - Wenn Objektstruktur viele Klassen von Objekten mit versch. Schnittstellen enthält und Operationen ausgeführt werden sollen, die von ihren konkreten Klassen abhängen
  - Mehrere versch. Operationen sollen auf Objekten einer Objektstruktur ausgeführt werden, Klassen sollen aber nicht mit diesen Operationen „verschmutzt" werden
  - Wenn Klassen, die eine Objektstruktur definieren, sich nie ändern, aber häufig neue Operationen für die Struktur definiert werden

3.2.3 Fabrikmethode (engl. factory method)

- ♥ Zweck:
  - Definiere Klassenschnittstelle mit Operationen zum Erzeugen eines Objekts, aber Unterklassen entscheiden von welcher Klasse das zu erzeugende Objekt ist
  - Delegiert Erzeugung von Objekten an Unterklassen
- ♥ Struktur:

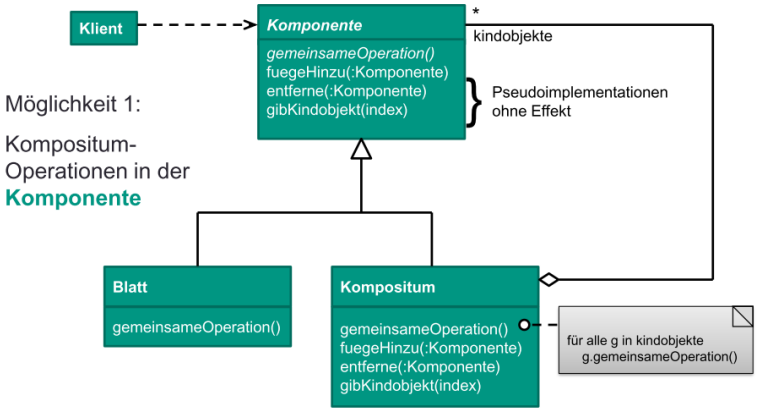


♥ Anwendbarkeit:
- Wenn die Klasse von zu erzeugenden Objekten nicht im Voraus bekannt ist
- Wenn Unterklasse die von der Oberklasse zu erzeugenden Objekte festlegen soll
- Fabrikmethode ist die Einschubmethode bei einer Schablonenmethode für Objekterzeugung

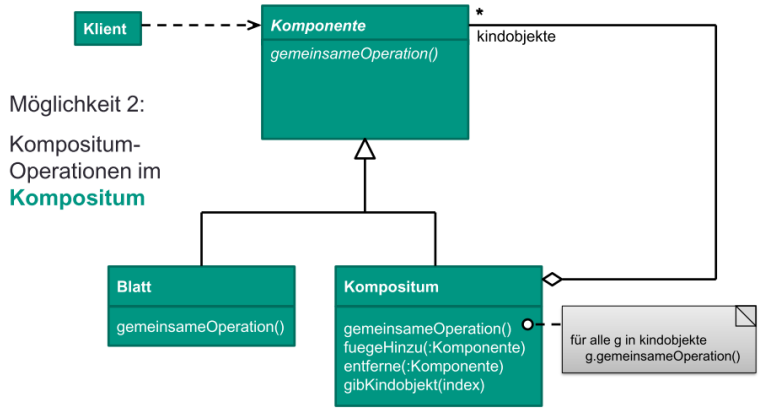
3.2.4 Kompositum (engl. composite)

♥ Zweck:
- Führt Objekte zu Baumstrukturen zusammen
- Repräsentiert Bestandshierarchien
- Ermöglicht es Klienten, Aggregate oder einzelne Objekte einheitlich zu behandeln

♥ Struktur:
- Möglichkeit 1:



- Möglichkeit 2:



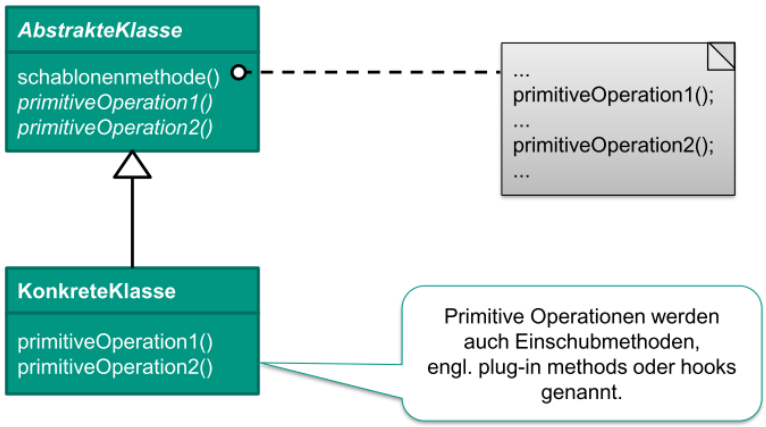
♥ Anwendbarkeit:
- Wenn Bestands-Hierarchien von Objekten repräsentiert werden sollen
- Wenn Klienten in der Lage sein sollen, Unterschiede zwischen einzelnen und zusammengesetzten Objekten zu ignorieren

♥ Implementierung:
- Nützlich, Eltern-Referenz in jeder Komponente zu führen
- Die Komponenten-Schnittstelle sollte so viele gemeinsame Methoden des Kompositums und der Blätter wie möglich definieren um Transparenz zu garantieren
- Wenn Methoden des Kompositums in der Komponente definiert werden, sollte gibKindobjekt() bei Blättern nichts zurückgeben
- fuegeHinzu() und entferne() sollten bei Blättern fehlschlagen und einen Fehler zurückgeben oder eine Ausnahme generieren
- Speichern der Kindelemente: Kinder müssen unter Umständen in einer bestimmten Reihenfolge

gelassen werden, Bei einer festen Anzahl Kinder verwende explizite Variablen und spezialisierte fuegeHinzu()/entferne()/gibKindobjekt() Operationen (zb setzeLinks())

3.2.5 Schablonenmethode (engl. template method)

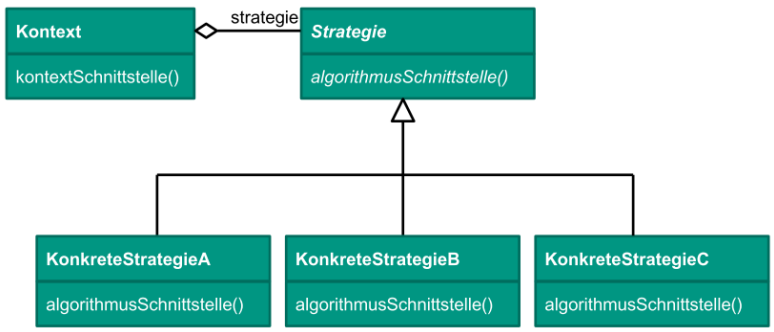
- ♥ Zweck:
  - Definiere Skelett eines Algorithmus in einer Operation, delegiere einzelne Schritte an Unterklassen
  - Ermöglicht es Unterklassen bestimmte Schritte eines Algorithmus zu überschreiben, ohne seine Struktur zu verändern
- ♥ Struktur:



- ♥ Anwendbarkeit:
  - Um invarianten Teile eines Algorithmus einmal festzulegen. Unterklassen implementieren das variierende Verhalten
  - Wenn gemeinsames Verhalten aus Unterklassen herausfaktorisiert und in gemeinsamer Klasse platziert werden soll um Code-Duplikate zu vermeiden
  - Um Erweiterungen mit Unterklassen zu kontrollieren: Ruft Einschubmethoden an bestimmten Stellen auf → Nur dort werden Erweiterungen zugelassen
  - Häufig bei Rahmenarchitekturen genutzt

3.2.6 Strategie

- ♥ Zweck:
  - Definiere Familie von Algorithmen, kapsle sie und mache sie austauschbar
  - Ermöglicht es, Algorithmus unabhängig von nutzenden Klienten zu variieren (Manchmal müssen Algorithmen, abhängig von der notwendigen Performanz, der Menge oder des Typs der Daten, variiert werden)
- ♥ Struktur:

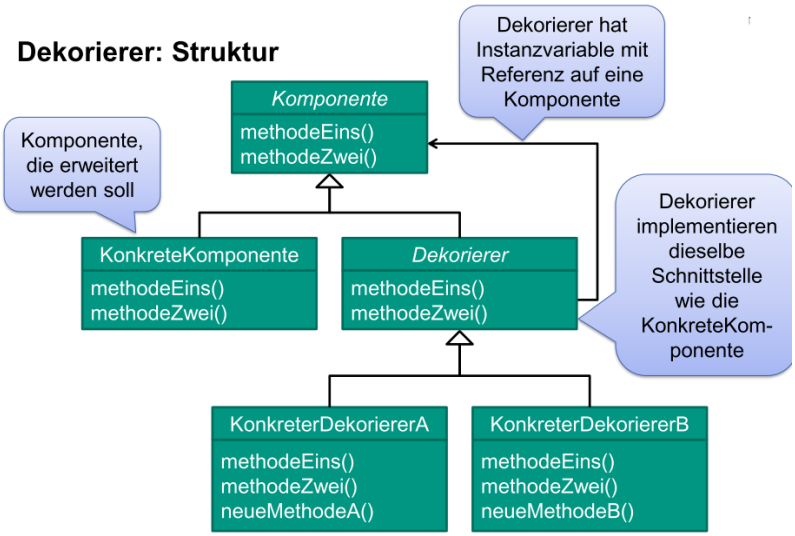


- ♥ Anwendbarkeit:
  - Wenn sich viele Klassen nur in ihrem Verhalten unterscheiden. Strategieobjekte ermöglichen Konfiguration einer Klasse mit einer von mehreren möglichen Verhaltensweisen
  - Wenn versch. Varianten eines Algorithmus benötigt werden

- Datenstrukturen des Algorithmus sollen vor Klienten versteckt werden
- Anzahl Fallunterscheidungen soll verringert werden
- Erlaubt dynamische Veränderung des Verhaltens

3.2.7 Dekorierer

- ♥ Zweck:
  - Fügt dynamisch neue Funktionalität zu Objekt hinzu
  - Alternative zu Vererbung
- ♥ Struktur:



- ♥ Dekorierer vs. Stellvertreter:

Dekorierer	Stellvertreter
- Fügt Objektfunktionalität hinzu, ohne Subjekt zu ändern - Kann Subjektschnittstelle erweitern	- Zugriffssteuerung - Kann genauso wie das Subjekt verwendet werden - Kann Latenz verstecken - Kann Methoden verstecken - Eigenes Objekt mit Subjekt "im Hintergrund"

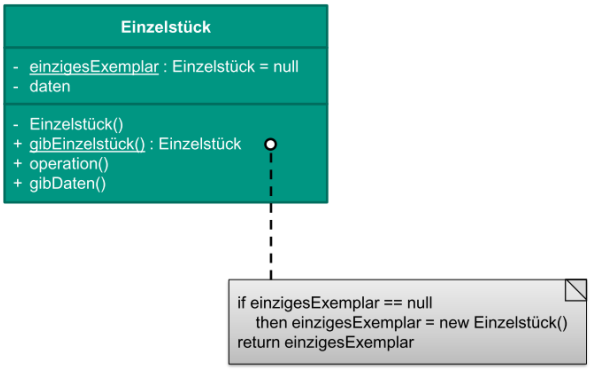
3.3 Zustandshabungs-Muster

- Bearbeiten den Zustand von Objekten, unabhängig von deren Zweck

3.3.1 Einzelstück (engl. singleton)

- ♥ Zweck:
  - Sichert zu, dass Klasse genau ein Exemplar besitzt
  - Globaler Zugriffspunkt auf das Exemplar
  - Klasse ist selbst für die Verwaltung ihres einzigen Exemplars zuständig
  - Klasse kann durch Abfangen von Befehlen zur Erzeugung neuer Objekte sicherstellen, dass kein weiteres Exemplar erzeugt wird

♥ Struktur:

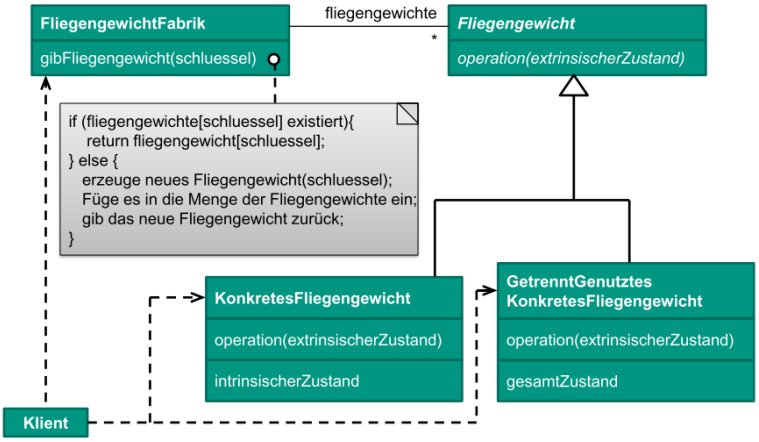


♥ Anwendbarkeit:
- Wenn es von Klasse nur eine Instanz geben darf und diese an bekannter Stelle zugänglich sein soll
- Wenn einzige Instanz durch Unterklassenbildung erweiterbar sein soll

3.3.2 Fliegengewicht (engl. flyweight)

♥ Zweck:
- Objekte kleinster Granularität gemeinsam nutzen, um große Mengen davon effizient speichern zu können

♥ Struktur:

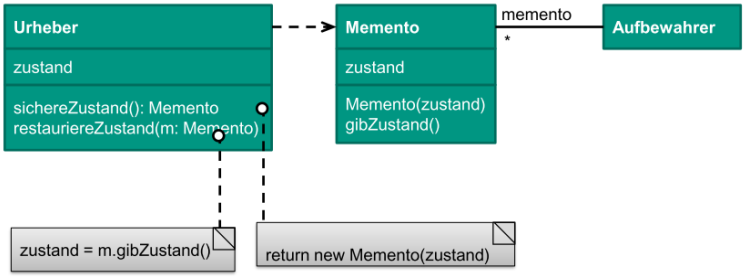


♥ Anwendbarkeit:
- Anwendung nutzt viele Objekte und
- Speicherkosten nur wegen Anzahl der Objekte hoch und
- Großteil des Objektzustands kann extrinsisch gemacht werden können und
- Anwendung hängt nicht von Identität der Objekte ab

3.3.3 Memento

- Zweck:
- Erfasst, externalisiert internen Zustand eines Objekts ohne Kapselung zu verletzen
- Objekt kann später in diesen Zustand zurückversetzt werden

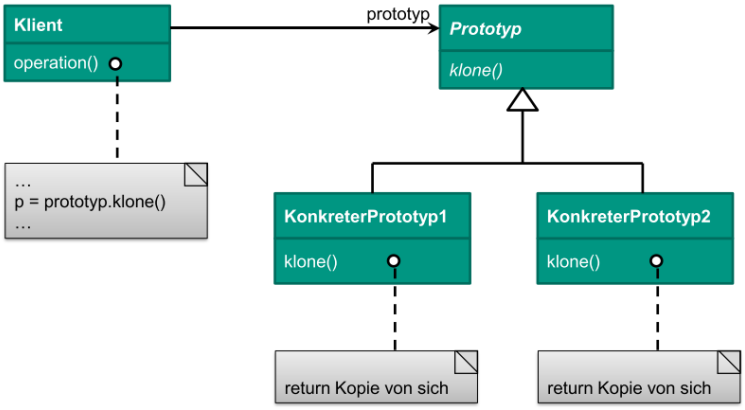
- Struktur:



- Anwendbarkeit:
  - Momentaufnahme eines Zustands eines Objekts soll zwischengespeichert werden, Objekt soll später in diesen Zustand zurückversetzt werden
  - Wenn direkte Schnittstelle zum Zustand Implementierungsdetails offenlegen / Kapselung aufbrechen würde

3.3.4 Prototyp

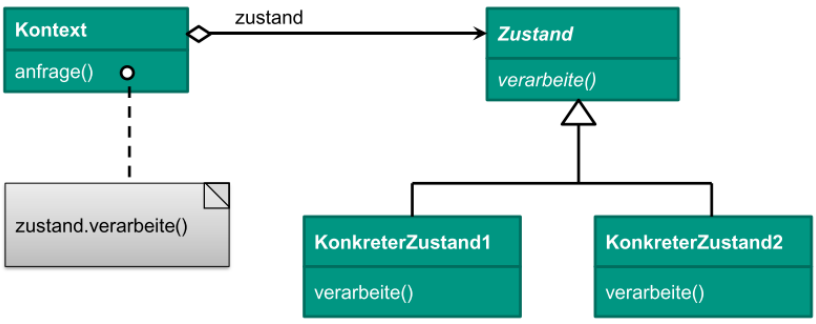
- ♥ Zweck:
  - Bestimmt Art der zu erzeugenden Objekte durch Verwendung eines Exemplars, Erzeugt neue Objekte durch Kopieren dieses Prototyps
- ♥ Struktur:



- ♥ Anwendbarkeit:
  - Wenn System unabhängig davon sein soll, wie seine Produkte erzeugt / repräsentiert werden sollen
  - Falls der Aufbau eines Objekts mehr Zeit braucht als Kopie anzulegen
  - Wenn Klassen zu erzeugender Objekte erst zur Laufzeit spezifiziert werden

3.3.5 Zustand (engl. state)

- ♥ Zweck:
  - Ändert Verhalten eines Objekts, wenn sich Zustand ändert
- ♥ Struktur:



♥ Anwendbarkeit:
- Wenn Verhalten eines Objekts vom Zustand abhängt, Objekt verändert sein Verhalten während der Laufzeit abhängig vom Zustand

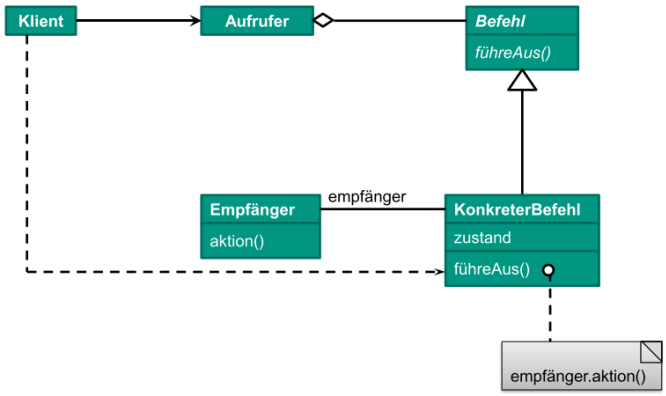
3.4 Steuerungs-Muster

- Steuern den Kontrollfluss
- Ergebnis: Richtige Methoden werden zur richtigen Zeit aufgerufen

3.4.1 Befehl

♥ Zweck:
- Befehl als Objekt kapseln
- Ermöglicht es, Klienten mit versch. Anfragen zu parametrisieren, Operationen in eine Warteschlange zu stellen, Operationen rückgängig machen

♥ Struktur:

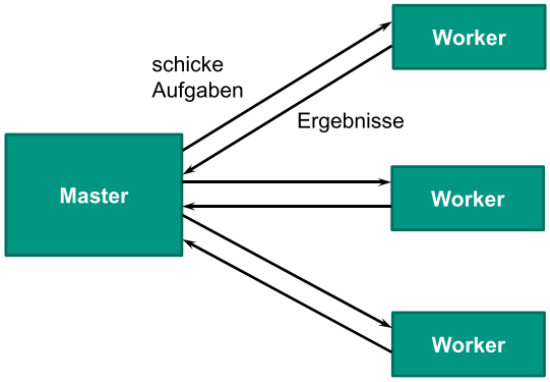


♥ Anwendbarkeit:
- Wenn Objekte mit auszuführender Aktion parametrisiert werden sollen
- Anfragen zu versch. Zeiten spezifiziert, aufgereiht, ausgeführt werden sollen
- Operationen sollen rückgängig gemacht werden
- System soll mittels komplexer Operationen strukturiert werden, die aus primitiven Operationen aufgebaut werden

3.4.2 Auftraggeber/ -nehmer (engl. master/worker)

♥ Zweck:
- Bietet fehlertolerante, parallele Berechnung
- Auftraggeber verteilt Arbeit an identische Arbeiter, berechnet Endergebnis aus Teilergebnissen der Arbeiter

♥ Struktur:



♥ Anwendbarkeit:
- Wenn mehrere Aufgaben bestehen, die unabhängig bearbeitet werden können
- Mehrere Prozessoren zur parallelen Verarbeitung sind vorhanden
- Belastung der Arbeiter soll ausgeglichen werden

3.5 Virtuelle Maschinen

- Eingabe: Daten, ein Programm
- Führen das Programm selbständig an den Daten aus
- In Software implementiert

3.6 Bequemlichkeitsmuster

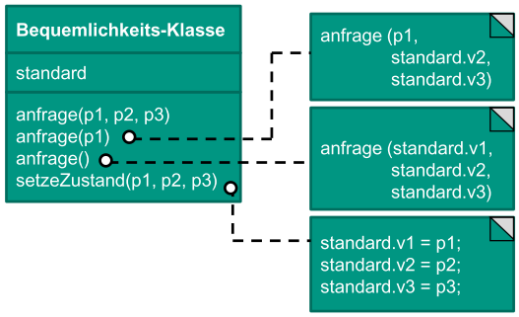
- Sparen Schreib- oder Denkarbeit

3.6.1 Bequemlichkeits-Klasse (engl. convenience class)

♥ Zweck:
- Vereinfacht Methodenaufruf durch Bereithaltung der Parameter in einer Klasse

♥ Anwendbarkeit:
- Wenn Methoden häufig mit gleichen Parametern aufgerufen werden

♥ Struktur:

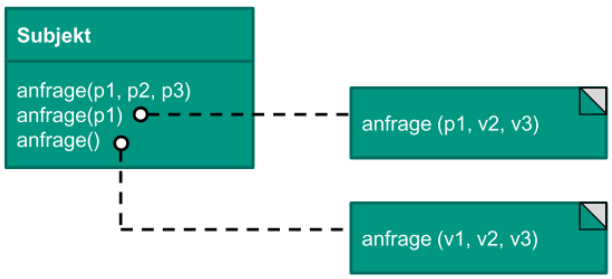


3.6.2 Bequemlichkeits-Methode (engl. convenience method)

♥ Zweck:
- Vereinfacht Methodenaufruf durch Bereithaltung häufiger Parameter in einer Methode / mehreren Methoden

♥ Anwendbarkeit:
- Wenn Methoden häufig mit den gleichen Parametern aufgerufen werden

♥ Struktur:



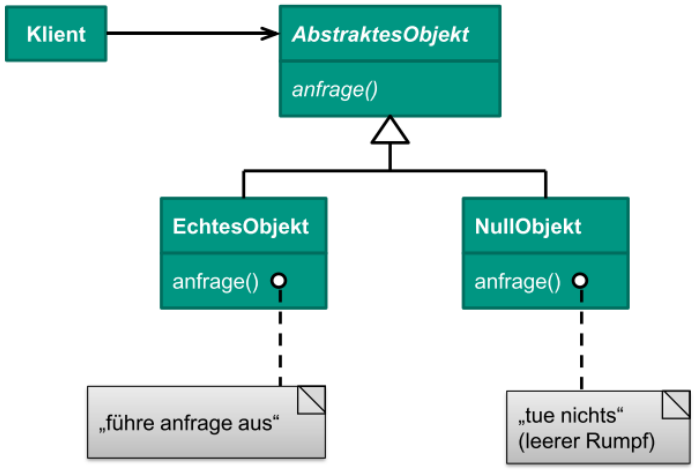
3.6.3 Fassade (engl. facade)

♥ Zweck:
- Bietet einheitliche Schnittstelle zu einer Menge von Schnittstellen eines Subsystems
- Definiert abstrakte Schnittstelle, Vereinfacht Benutzung des Subsystems

- ♥ Anwendbarkeit:
  - Einfache Schnittstelle zu komplexen Subsystem soll angeboten werden
  - Wenn es viele Abhängigkeiten zwischen Klienten und Implementierungsklassen einer Abstraktion gibt
  - Wenn Subsysteme in Schichten aufgeteilt werden sollen (Fassade als Eintrittspunkt einer Schicht)

3.6.4 Null-Objekt (engl. null object)

- ♥ Zweck:
  - Stellt Stellvertreter zur Verfügung, der gleiche Schnittstelle bietet aber nichts tut
  - Kapselt Implementierungsentscheidungen (Wie tut es nichts)
  - Verhindert, dass Code mit Tests gegen null-Werte verschmutzt wird
- ♥ Struktur:



- ♥ Anwendbarkeit:
  - Wenn Objekt Mitarbeiter benötigt, von denen mind. 1 nichts tun soll
  - Klienten sollen sich nicht um Unterschied zwischen echtem Mitarbeiter und einem der nichts tut kümmern
  - „Tue nichts“ soll von versch. Klienten wiederverwendet werden