

Entwurfsmuster beschreibt Familie von Lösungen für SW-Entwurfsproblem

Entwurfsmuster für Entwurf \Leftrightarrow Algorithmen für Programmieren

(1) - Verbessern Kommunikation im Team

- Diskussion über komplexe Konzepte
- erfassen wesentliche Konzepte, bringen sie in verständliche Form
 - helfen Entwürfe zu verstehen
 - dokumentieren Entwürfe kuri und knapp
- können Code-Qualität, -struktur verbessern

Kategorien:

(1) Entkopplungsmuster

teilen System in mehrere Einheiten, einzelne Einheiten können unabh. erstellt, verändert, ausgetauscht, wieder verwendet werden \rightarrow System kann durch lokale Änderungen verbessert, ... werden ohne ganzes System modifiziert

Adapter
Beobachter
Brücke
Herautor
stellvertretender
Vermittler

(2) Varianten-Muster

Gemeinsamkeiten aus verwandten Einheiten rausziehen, an einer einzigen Stelle beschreiben \rightarrow vermeidet Redundanz

Abstrakte Fabrik
Besucher
(Erbauer)
Fabrikmethode

Kompositum
Schablonenmethode
Strategie
Dekomponer

(3) Zustandsübertragungsmuster

bearbeiten Zustand der von Objekten, unabh. von deren Zweck

Einzelstück
Fliegengewicht
Prototyp

Memento
Zustand

(4) Steuerungsmuster

steuern Kontrollfluss, bewirken, dass zur richtigen Zeit die richtigen Methoden aufgerufen werden

Befehl
Master/Worker

(5) Virtuelle Maschinen

erhalten Daten + Programm als Eingabe, führen Programm selbstständig an Daten aus

(interpretierend)

(6) Bequemlichkeitsmuster

sparen Schreib-/Denkarbeit

Bequemlichkeits-Klasse
Bequemlichkeits-Methode

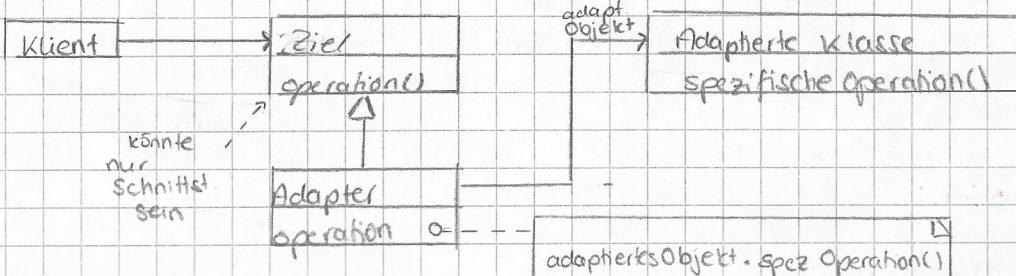
Fassade
Null-Objekt

Entkopplungsmuster

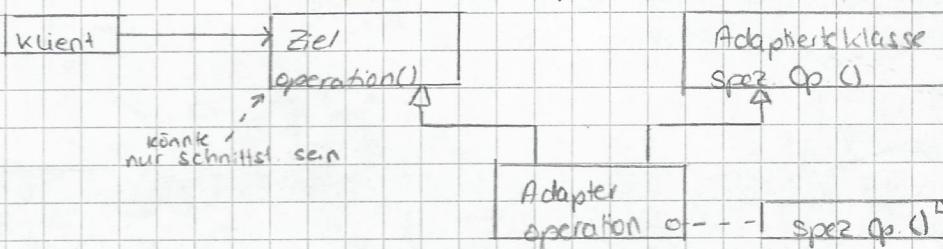
Adapter (Object Adapter)

- > passt Schnittstelle einer Klasse an von Klienten erwartete Schnittstelle an
- ↳ lässt Klassen zusammenarbeiten, die eig. inkompatible Schnittstellen haben

Objektaadapter: ohne Mehrfachvererbung



Klassenadapter: mit Mehrfachvererbung

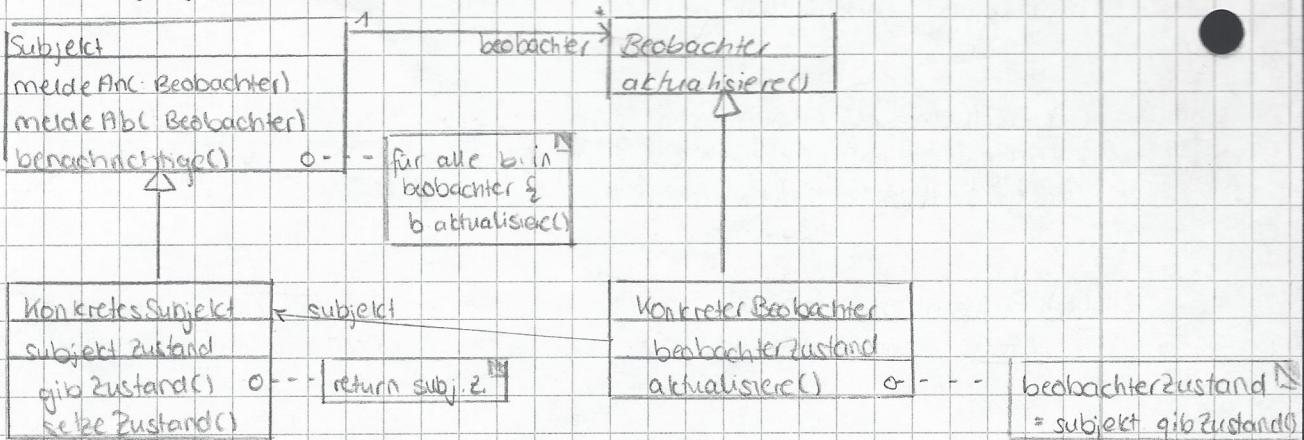


Anwendbarkeit:

- verwende ex. Klasse, aber Schnittstelle passt nicht (und kann nicht geändert werden)
- Erstellung wiederverwendbarer Klassen, die mit unvorhersehbaren Klassen zusammenarbeiten
(↳ evtl. haben diese keine kompatible Schnittstelle)
- Verwendung ex. Unterklassen ohne alle Schnittst. anzupassen → Objektaadapter

Beobachter (Observer):

1-zu-n-Abh. zw. Objekten: Änderung eines Zustands eines Objektes führt zu Benachrichtigung + automatischer Aktualisierung aller abh. Objekte

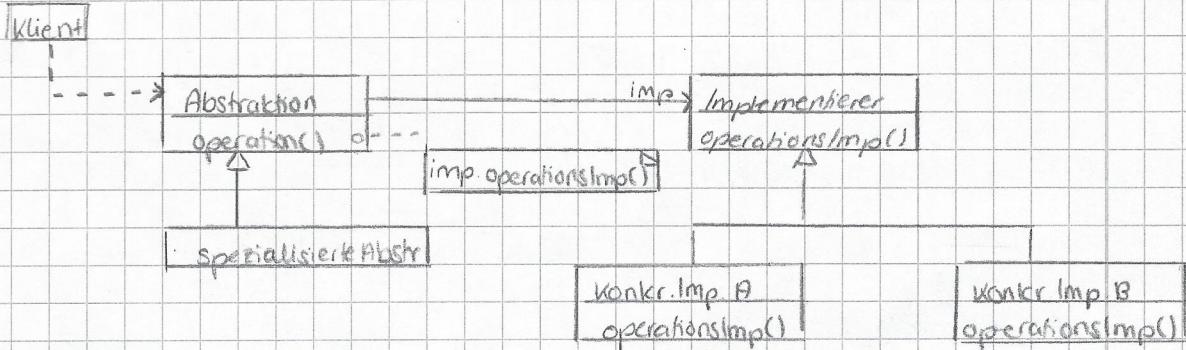


Anwendbarkeit:

- Änderung eines Objekts verlangt Änderung anderer Objekte (unbekannt welche/wieviele)
- Objekt muss andere benachrichtigen ohne Annahmen über diese zu treffen

Brücke (bridge):

Entkoppeln Abstraktion von ihrer Implementierung
 ↳ können unabh. voneinander variiert werden

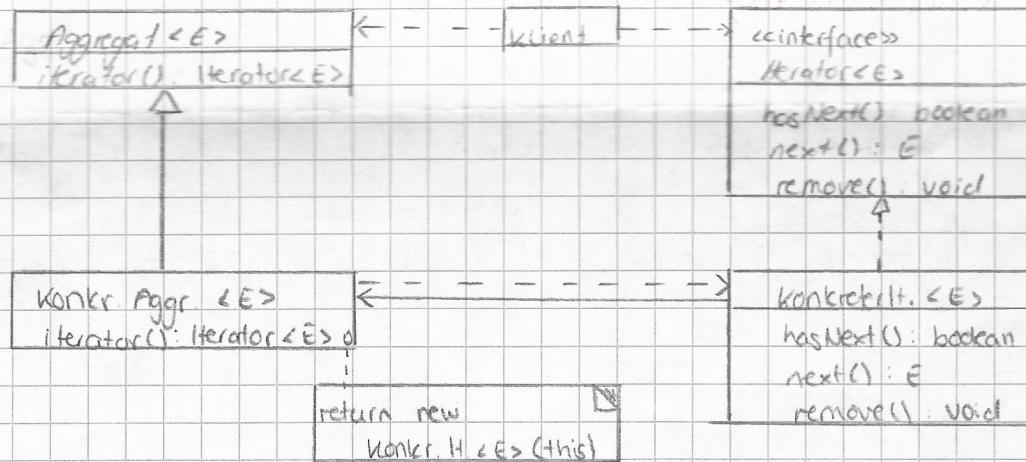


Anwendbarkeit:

- Vermeidung dauerhafter Verbindung zw. Abstraktion und Implementierung
- Abstraktion und Implementierung sollen durch Unterklassen erweiterbar sein
- Änderungen in Impl. der Abstr. sollen keine Auswirkung auf Klienten haben
- Impl. einer Abstraktion soll vollständig vom Klienten verdeckt werden
- Impl. soll von mehreren Obj. gemeinsam benutzt werden

Iterator (iterator):

ermöglicht sequentiellen Zugriff auf Element eines zusammengesetzten Objekts, ohne zugrundeliegende Repräsentation offen zu legen

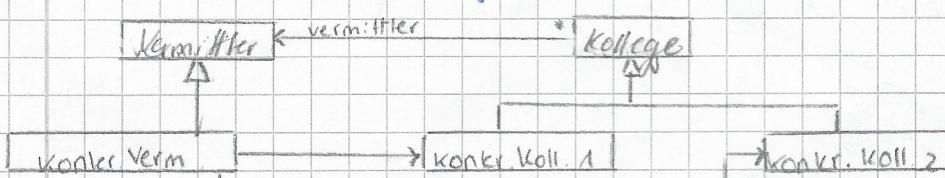


Anwendbarkeit:

- ermögliche Zugriff auf Inhalt eines zusammengesetzten Objekts ohne interne Struktur offen zu legen
- einheitliche Schnittstelle zur Traversierung untersch. zusammengesetzter Strukturen
- ermöglicht gleichzeitig mehrere Traversierungen → Robust
 - Robust wenn jede Instanz des Iterators zu einer Datenstruktur eigenen Zeiter besitzt
 - ↳ versch. Instanzen können unabh. benutzt werden

Vermittler (mediator):

Definiert Obj., welche Zusammenspiel einer Menge von Obj. kapselt
 ↳ fördern lose Kopplung: Obj. nehmen nicht explizit Bezug aufeinander

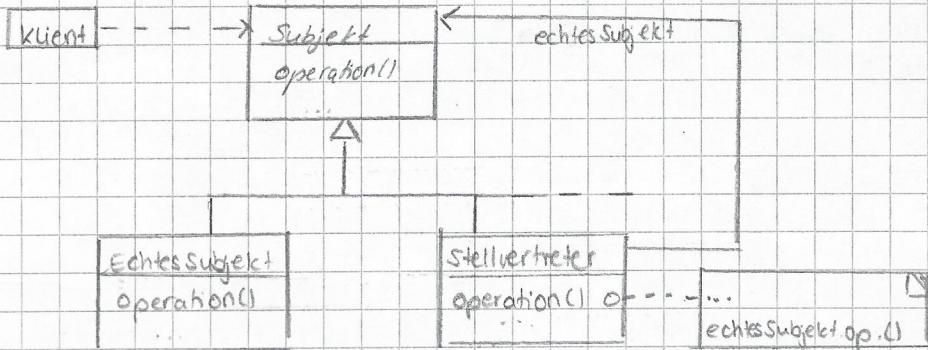


Anwendbarkeit:

- wenn Menge von Obj. in wohldef., aber komplexer Weise zusammenarbeiten → Übersicht
- Wiederverwendung eines Obj., was sich auf andere Obj. bezieht vereinfaches
- auf mehrere Klassen verteiltes Verhalten soll maßgeschneidert werden, ohne viele Unterklassen

Stellvertreter (proxy)

Kontrollierte Objektzugriff mit vorgelegtem Stellvertreterobjekt



Anwendbarkeit

- Bei Bedarf nach anpassungsfähiger, intelligenter Referenz auf Objekt als Zeiger
- protokollierender Stellvertreter: zählt Referenzen auf eig. Obj.

↳ keine Ref. ex. mehr: Obj. wird freigegeben

caching proxy - puffernder Stellv.: lädt persistentes Obj. erst dann in Speicher, wenn das erste Mal angefordert

remote proxy - Fernzugriffsvetretner: lokaler Stellv. für Obj. in anderem Adressraum

- Platzhalter (virtual Proxy): erzeugt teure Obj. auf Verlangen (verzögertes Laden/Erzeugen)

- Firewall: kontrolliert Zugriff auf Originalobj. (z.B. wenn Obj. versch. Zugriffsrechte haben)

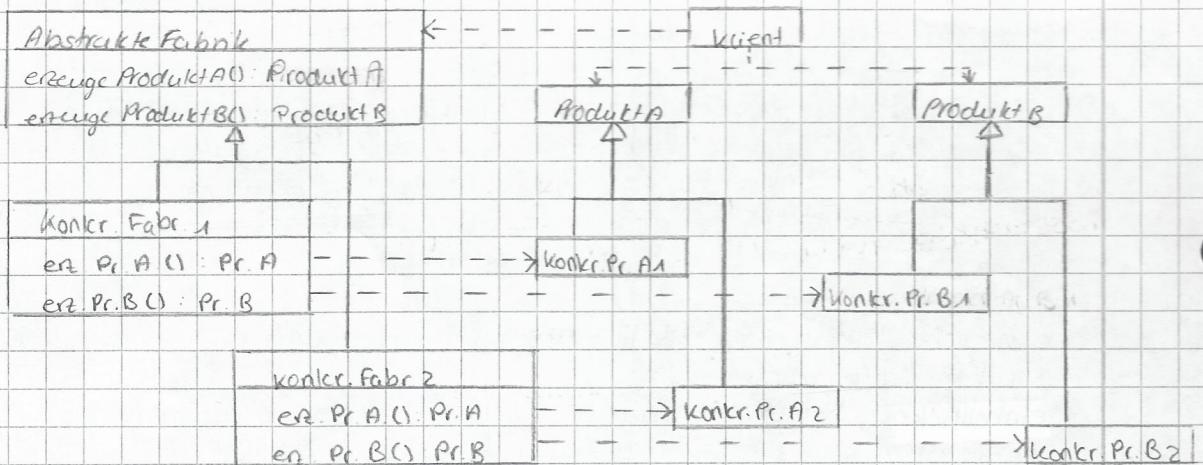
- Synchronisationsvertretner: coord. Zugriff mehrerer Threads auf ein Obj.

- Detonierer fügt zusätzl. Zuständigkeiten zu bestehendem Obj. hinzu

Varianten - Muster

Abstrakte Fabrik (abstract factory)

Schnittstelle zum Erzeugen von Familien verwandter, voneinander unabh. Obj., ohne ihre konkreten Klassen zu benennen



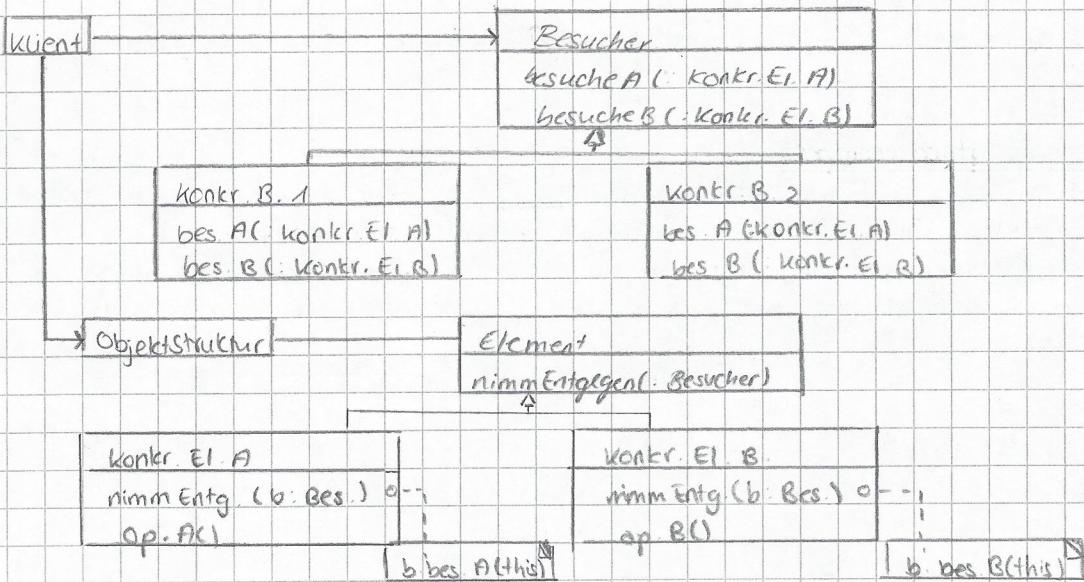
Anwendbarkeit:

- System soll unabh. von Produktierung, -zusammensetzung, -repräsentation sein
- System soll mit einer von mehreren Produktfamilien konfiguriert werden
- Fam. von aufeinander abgestimmten Produktobj. soll zusammen verwendet werden, ohne dies zu erzwingen
- Klassenbib., die nur Schnittst., nicht aber impl. offen legt

3

Besucher (visitor)

auf Elementen einer Obj. struktur auszuführende Op. als Objekt kapseln
 → ermöglicht Def. einer neuen Op., ohne Klassen der von ihr bearbeiteten Elemente zu verändern

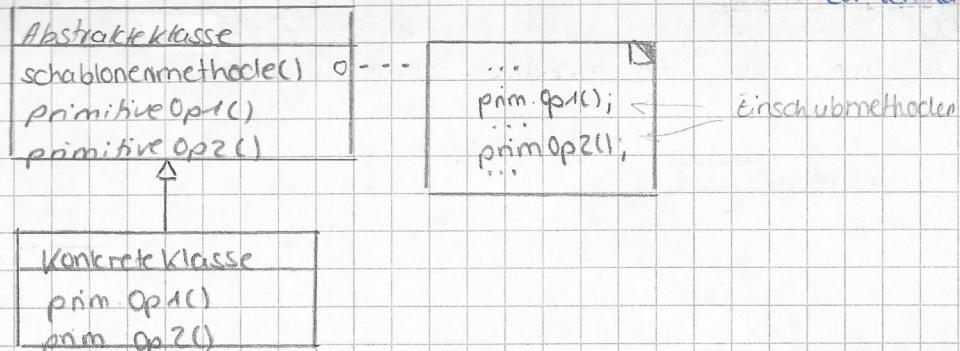


Anwendbarkeit:

- Obj. struktur enth. viele Klassen von Obj. mit untersch. Schnittst. , auszuführende Op. hängen von ihren konkreten Klassen ab
- mehrere untersch. nicht verwandte Op. auf Obj. einer Obj.struktur ausführen , ohne diese zu "vermischen"
- Klassen einer Obj.struktur ändern sich (fast) nie, aber oft neue Op. auf Struktur

Schablonenmethode (template method)

definiere Stellst eines Algorithmus in einer Op., delegiere einzelne Schritte an Unterklassen
 → Unterklassen können bestimmte Schritte eines Algos überschreiben, ohne Struktur zu verändern

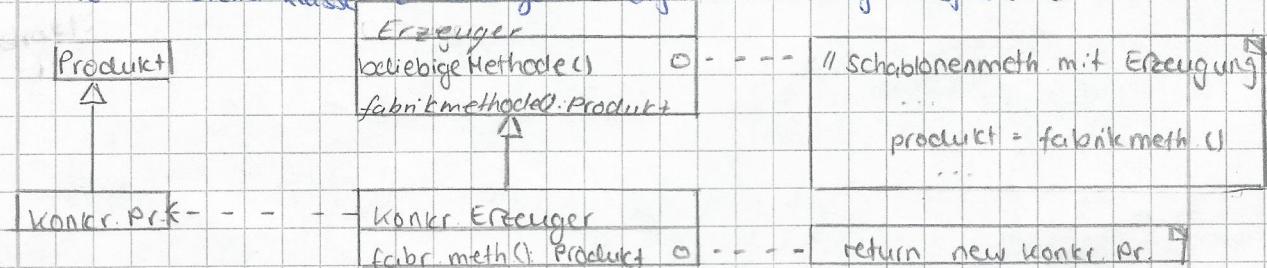


Anwendbarkeit:

- lege invarianten Teil des Algos genau einmal fest, Unterklassen impl. variierendes Verhalten
- faktorisierte gemeinsames Verhalten aus Unterklassen heraus → vermeidet Redundanz
- kontrolliert Erweiterungen durch Unterklassen: erlaubt Erweiterungen nur an bestimmten Stellen der Schablonenmeth. durch Einschubmethoden

Fabrikmethode (factory method)

def. Klassenschnittst. mit Op. zur Objekterzeugung, lasse Unterkl. entscheiden,
 von welcher Klasse zu erzeugendes Obj. ist → delegiere Obj.erz. an Unterkl.



Anwendbarkeit:

- eine Klasse kennt Klasse von Obj., die sie erzeugen muss nicht im Voraus
- Unterkl. soll erzeugende Obj. festlegen
- Klasse soll Zuständigkeiten an Hilfsunterkl. delegieren, wissen an welche soll lokalisert werden
- Fahrt. meth. \approx Einschubmethode bei Schabl. meth. für Objekterzeugung

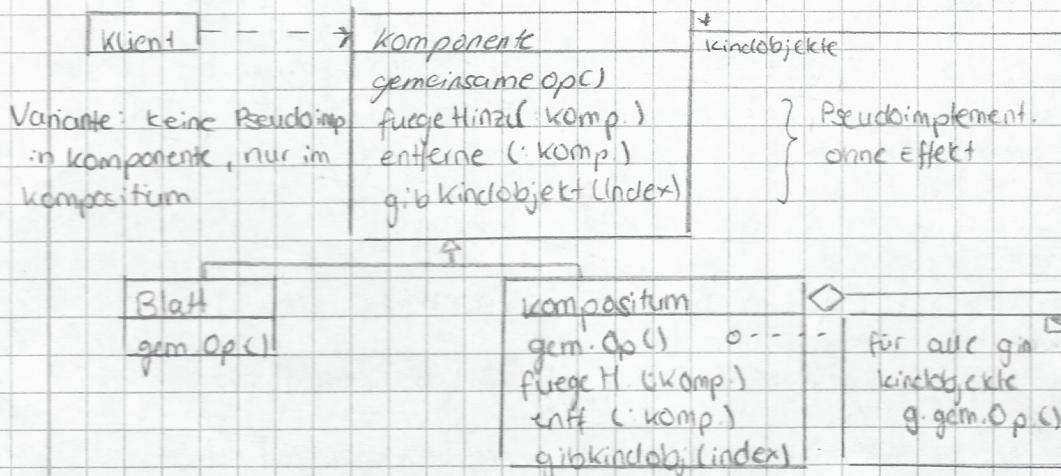
Kompositum (composite)

Füge Obj. zu Baumstrukturen zusammen, um Bestands hierarchien zu repräsentieren

\hookrightarrow Klient kann einzelne Obj., Agregate einheitlich behandeln

\Rightarrow fasse einfache Obj. zu Gruppen zusammen, diese zu größeren Gruppen, ...

\hookrightarrow Kompositum ist eine gemeinsame Eigenschaften und bildet daraus Oberklasse



Anwendbarkeit:

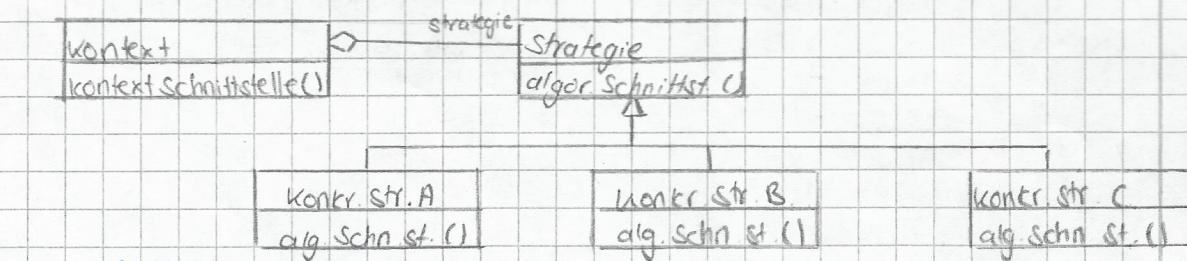
- Repräsentation von Bestands hierarchien von Obj.

- Klient soll in der Lage sein untersch. zw. zusammengesetzten und einzelnen Obj. zu ignorieren

Strategie (strategy)

Def. Fam. von Algos, kapsle sie und mache sie austauschbar

\hookrightarrow kann unabh. von nutzenden Klienten variiert werden



Anwendbarkeit:

- viele verwandte Klassen unterscheiden sich nur im Verhalten

- untersch. Varianten eines Algos benötigt

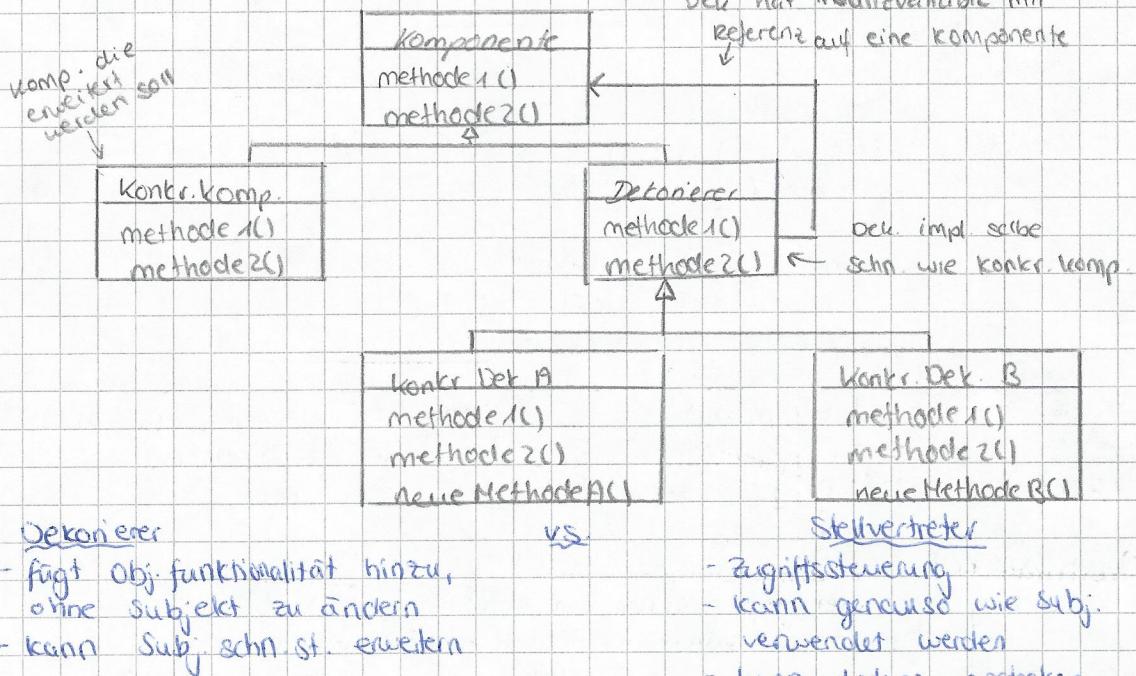
- Algo verwendet Datenstrukturen, die Klienten nicht bekannt sein soll

- Klasse def. untersch. Verhaltensweisen \rightarrow Strategie statt Fallunterscheidung

Dekomponierer (Decorator)

- Fügt dynamisch neue Funktionalität zu Obj. hinzu
- ↳ Alternative zur Vererbung

! nicht mit proxy verwechseln!



Zustands handhabungsmuster

Einzelstück (singleton)

scheze zu, dass Klasse genau ein Exemplar besitzt und stelle globalen Zugriffspunkt darauf bereit

Einzelstück

- einziges Exemplar: Einzelstück = null
- daten
- Einzelstück()
- + gibEinzelstück(): Einzelstück
- + operation()
- + gibDaten()

```

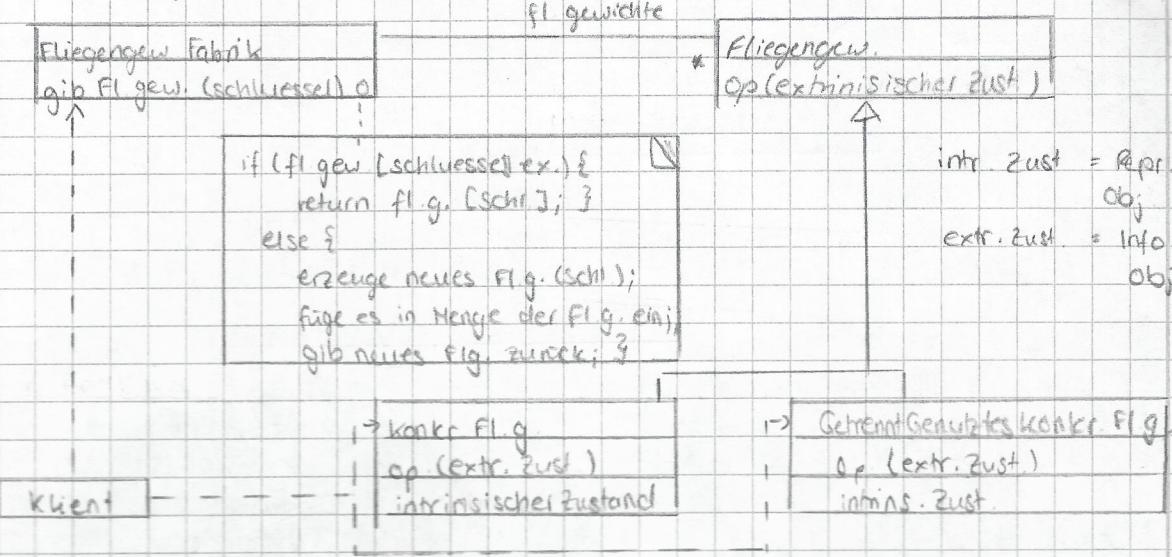
if einzigesExemplar == null
  then einz.Ex. = new Einzelst.()
return einz. Ex.
  
```

Anwendbarkeit:

- es darf nur eine Instanz einer Klasse geben, diese soll Klienten an bekannter Stelle zugängl. gemacht werden
- wenn es schwierig/unmöglich ist, festzustellen, welcher Teil der Anwendung die erste Instanz erzeugt
- wenn einzige Instanz durch Unterklassierung erweiterbar sein soll und Klienten diese ohne Veränderung ihres Quellkod. nutzen sollen

Fliegengewicht (flyweight)

nutzt Obj. kleinster Granularität gemeinsam, um große Mengen von ihnen effizient speichern zu können



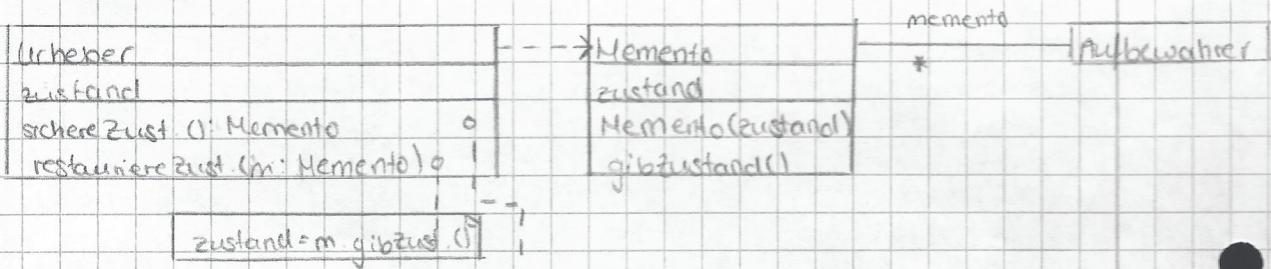
- intr. Zust. = repr. einzelner Obj. durch cycle
- extr. Zust. = Info/Op dieser Objekte

Anwendbarkeit:

- Anwendung verwendet große Menge von Objekten → hohe Speicher Kosten
 - wenn Großteil des Obj. Zustands extrinsisch gemacht werden kann
 - viele Gruppen von Obj. sollen durch relativ wenig gemeinsam genutzte Obj. ersetzt werden, sobald extrins. Zust. entfernt wurde und Anw. nicht von Identität der Obj. abhängt.

Memento (memento)

erfasse und externalisiere inneren Zustand eines Objekts, ohne seine Kapselung zu verlieren, sodass das Obj. später in diesen Zustand zurück versetzt werden kann

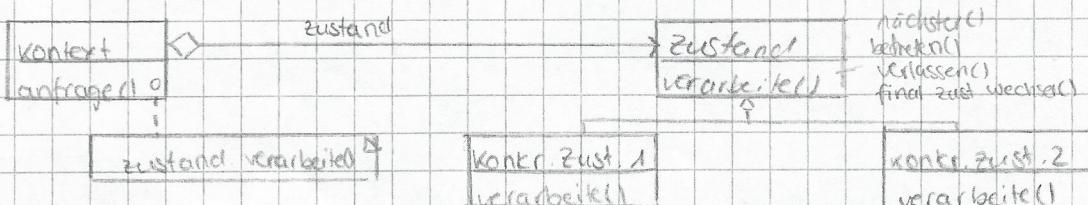


Anwendbarkeit:

- wenn Momentaufnahme des Zustands eines Obj. zwischengespeichert werden muss, so dass Obj. zu späterem Zeitpunkt darin in diesen zurückversetzt werden kann
 - wenn direkte Schnittst. zum Ermitteln des Zustands impl. Details öffneten würde

Zustand (state)

Ändere Verhalten des Obj., wenn sich interner Zustand ändert

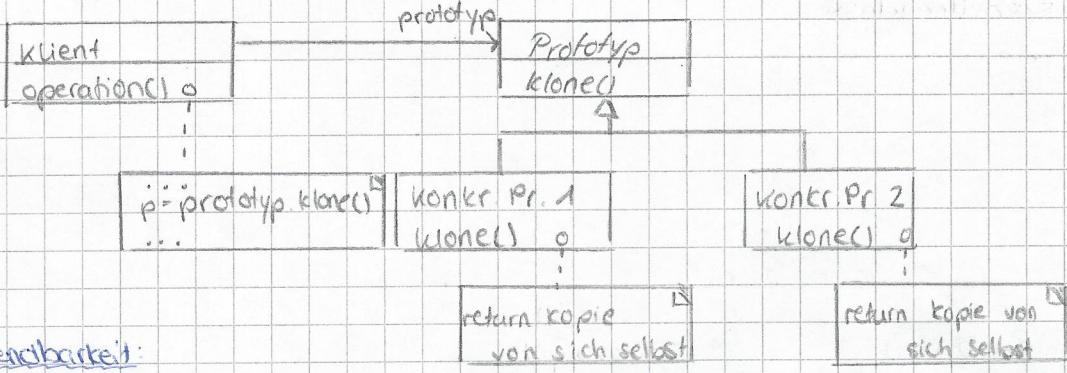


Anwendbarkeit:

- Verhalten des Obj. abhängig vom Zustand, Obj. muss Verhalten abh. vom aktuellen Zustand ändern

Prototyp (prototype)

Bestimme die Arten zu erzeugender Objekt durch die Verwendung eines typischen Exemplars und erzeuge neue Obj. durch Kopieren dieses Prototyps



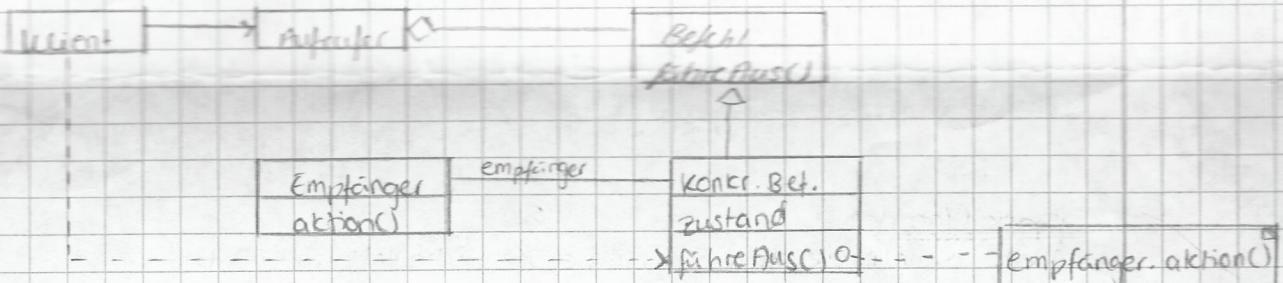
Anwendbarkeit:

- System soll unabh. davon sein, wie seine Produkte erstellt, zusammengesetzt, repr. werden
 - falls Aufbau eines Obj. wesentlich mehr Zeit braucht, als Kopie anstreifen
 - falls Klassen zu erzeugender Obj. erst zur Laufzeit spezifiziert werden
 - um Klassenhierarchie von Fabriken zu vermeiden, die parallel zur L.t.h. von Produkten verläuft

Steuerungsmuster

Befehl (command)

Kapsle Befehl als Obj. → parametrisierte Klienten mit versch. Anfragen, stelle Op. in Warteschlange, führe Logbuch, mache Op. rückgängig

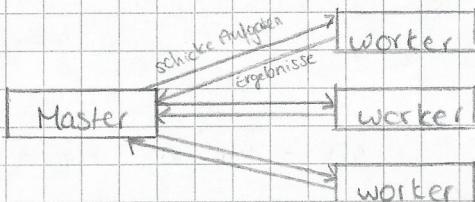


Anwendbarkeit:

- Obj. mit auszuführender Aktion parametrieren
 - Anfragen zu untersch. Zeiten spezifizieren, aufrufen, ausführen
 - Rückgängig machen von Op. (undo) soll unterstützt werden
 - Mitprotokollieren von Änderungen soll unterstützt werden

Auftraggeber/-nehmer (master/worker)

bietet fehlertolerante, parallele Berechnung; master verteilt Arbeiten identischer Worker, berechnet Ergebnis aus Teilergebnissen von Worker



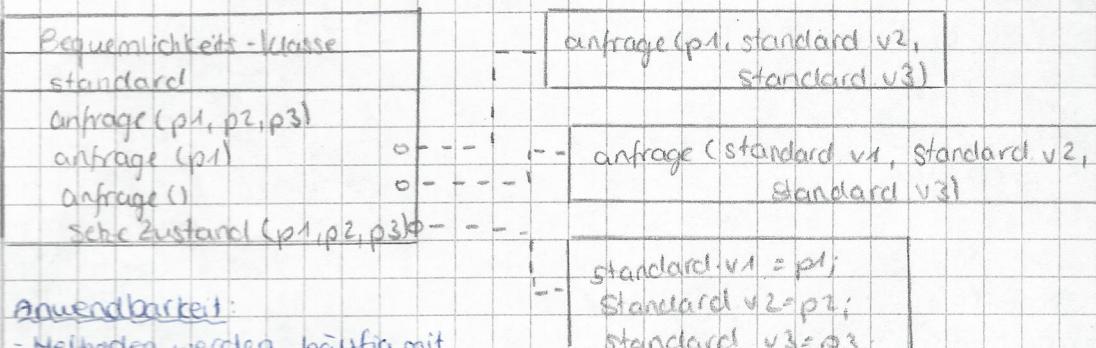
Anwendbarkeit:

- mehrere Aufg. können unabh. bearbeitet werden
 - mehrere Prozessoren zur parallelen Verarbeitung stehen zur Verfügung
 - Belastung der Arbeiter soll ausgeglichen werden

Bequemlichkeitsmuster

Bequemlichkeits-klasse (convenience class)

vereinfache Methodenaufruf durch Bereithaltung der Parameter in spezieller Klasse

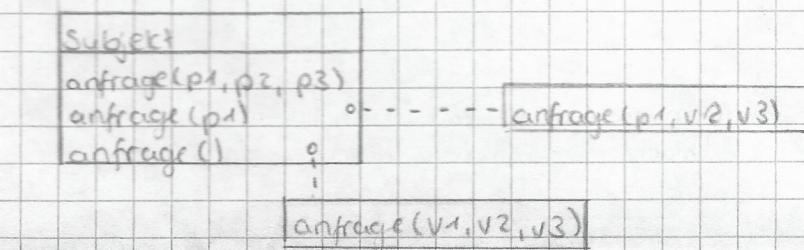


Anwendbarkeit:

- Methoden werden häufig mit gleichen Parametern aufgerufen, die sich nur selten ändern

Bequemlichkeits - Methode (convenience method)

vereinfachen des Methodenaufrufs durch Bereitstellung häufig genutzter Parameterkombinationen in zusätzlichen Methoden (Überladen)



Anwendbarkeit:

- Methodenaufrufe häufig mit gleichen Parametern

Fassade (facade)

einheitl. Schnittst. zu einer Menge von Schnittst. eines Subsystems
↳ definiert abstrakte Schnittstelle, welche Benutzung des Subs. vereinfacht

siehe vorne

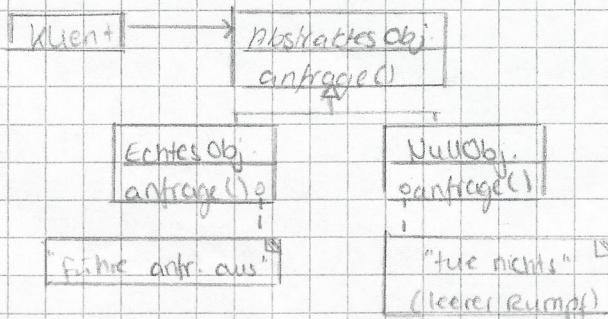


Anwendbarkeit:

- Direkt einfache Schnittst. zu komplexem Subs. an → Fassade
- viele Abh. zw. Klienten und Impl. Klassen einer Abstraktion
↳ Fassade entkoppelt Subs. von Klienten und anderen Subs.
- Einteilung der Subs. in Schichten → Fassade als Eingangspunkt

Null-Objekt (null object)

Stellvertreter mit gleicher Schnittstelle, der nichts tut → Null-Obj. kapselt Impl.-Entscheidungen



Anwendbarkeit:

- Obj. benötigt Mitarbeiter, einer/mehrere sollen nichts tun
- Klienten sollen untersch. zw. echtem Mitarb. und null obj. nicht interessieren
- "tut-nichts"-Verhalten soll von versch. Klienten verwendet werden