5.4 Entwurfsmuster

- auf Flexibilität und Wartungsfreundlichkeit zielende Systeme von Klassen
- Wiederverwendung auf Entwurfsebene
- jedes Muster hat Vor- und Nachteile, z.B. Flexibilität ↔ Overhead, # Objekte
- ein Problem z.T. durch verschiedene, alternative Muster lösbar
- für jede Anwendung überlegen, welches Muster die Ziele am besten erreicht
- Muster kombinierbar
- klassenbasiert ↔ objektbasiert
- Vererbung ↔ Delegation

Einteilung von Entwurfsmustern

1) Erzeugungsmuster:

   z.B. Abstrakte Fabrik, Erbauer, Prototyp, Singleton

2) Strukturmuster:

   z.B. Adapter, Brücke, Fassade, Fliegengewicht, Kompositum, Stellvertreter

3) Verhaltensmuster:

   z.B. Befehl, Beobachter, Besucher, Iterator, Memento, Strategie, Interpreter, Schablonenmethode, Vermittler, Zustand, Zuständigkeitskette

5.4.1 Erzeugungsmuster

- versteckt Erzeugungsprozess von Objekten
- System unabhängig von der Art der Erzeugung, Zusammensetzung bzw. Repräsentation der Objekte

5.4.1.1 Abstrakte Fabrik (Abstract Factory, Kit)

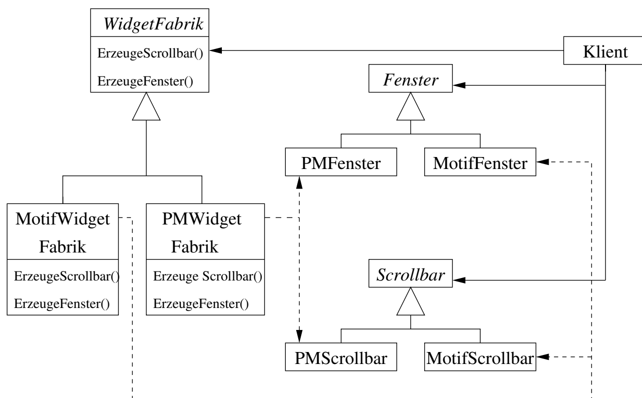
- objektbasiert

Ziel:

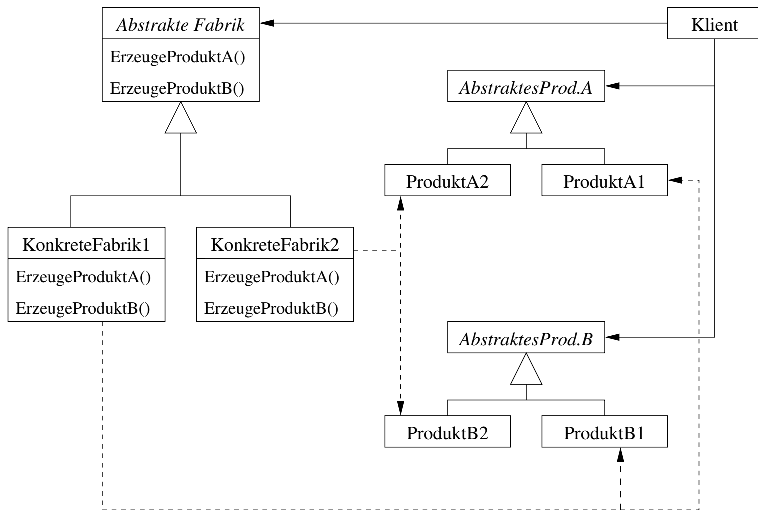
- bietet Schnittstelle zum Erzeugen von Familien verwandter Objekte, ohne ihre konkreten Klassen zu benennen

Abstrakte Fabrik: Beispielanwendung

- Klassenbibliothek für Benutzerschnittstellen, die mehrere Look&Feel-Standards unterstützen
  z.B. Motif, Presentation Manager



Abstrakte Fabrik: Allgemeine Struktur



Einordnung der Abstrakten Fabrik

Anwendbarkeit:

- wenn Unabhängigkeit von “Produkten” benötigt
- System soll mit Produktfamilie (→ Singleton) konfiguriert werden

Eigenschaften:

- Austausch von Produktfamilien durch eine einzige Änderung: den Austausch der konkreten Fabrik
- Unterstützung einer Produktfamillie mit anderer Schnittstelle schwierig

5.4.1.2 Fabrikmethode (Factory Method)

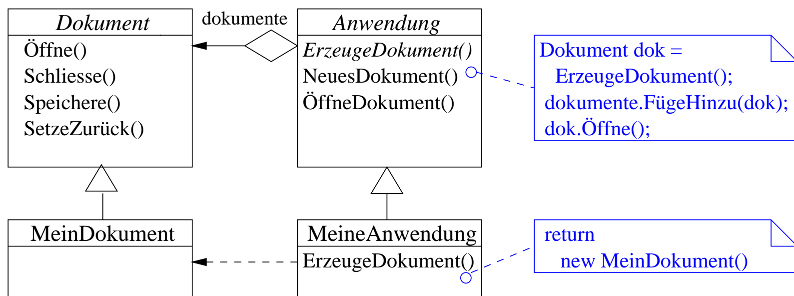
- klassenbasiertes Erzeugungsmuster

Ziel:

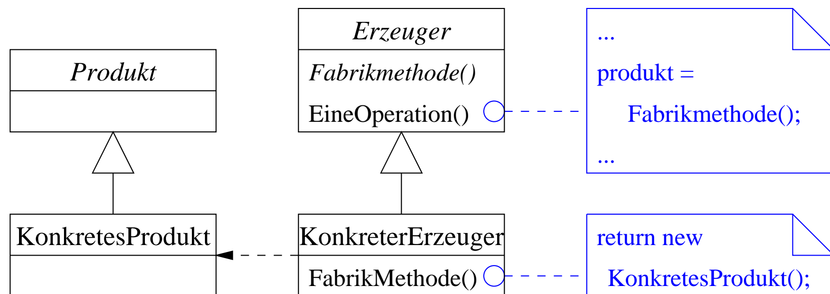
- ein Klasse (z.B. in Framework) muss Objekte erzeugen, deren Klasse sie nicht kennt
- Klassenschnittstelle mit Operationen zum Erzeugen eines Objekts
- Unterklassen entscheiden, von welcher Klasse das erzeugte Objekt ist

Fabrikmethode: Beispielanwendung

- Framework für Anwendungen,
  die mehrere Dokumente gleichzeitig präsentieren können
- Klassen `Anwendung` und `Dokument` abstrakt
- Klasse `Anwendung` muss Dokumente erzeugen, weiß aber nicht welche



Fabrikmethode: Allgemeine Struktur



Einordnung von Fabrikmethoden

Anwendbarkeit:

- eine Klasse kennt die Klassen der zu erzeugenden Objekte nicht

Eigenschaften

- Framework unabhängig von konkreter Anwendung
- Erzeugerklasse ggf. auch konkret
- Aufblähung der Klassenhierarchie
- default-Erzeugungsmuster; wird durch komplexere Muster ersetzt, wenn mehr Flexibilität nötig

Verwandte Muster

- die Abstrakte Fabrik wird oft durch Fabrikmethode implementiert

5.4.1.3 Singleton

- objektbasiertes Erzeugungsmuster

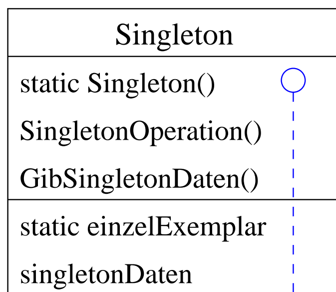
Ziel

- Klasse mit genau einem Objekt (vgl. globale Variable)
- Erzeugung weiterer Objekte wird abgefangen
- Klasse bietet einfachen Zugriff auf Einzelobjekt

Beispielanwendung

- zur Konfiguration des Systems z.B. beim Abstrakte-Fabrik- oder Erbauer-Muster

Singleton: Allgemeine Struktur und Einordnung



Anwendbarkeit

- wenn genau ein Exemplar einer Klasse erforderlich
- Exemplar “global” zugreifbar

Eigenschaften

- Zugriffskontrolle auf Einzelexemplar
- im Gegensatz zu globalen Variablen: strukturierter Adressraum z.B.`singleton1.a()`, `singleton1.b()` statt `a()`,`b()`

5.4.2 Strukturmuster

- zur Komposition von Klassen und Objekten

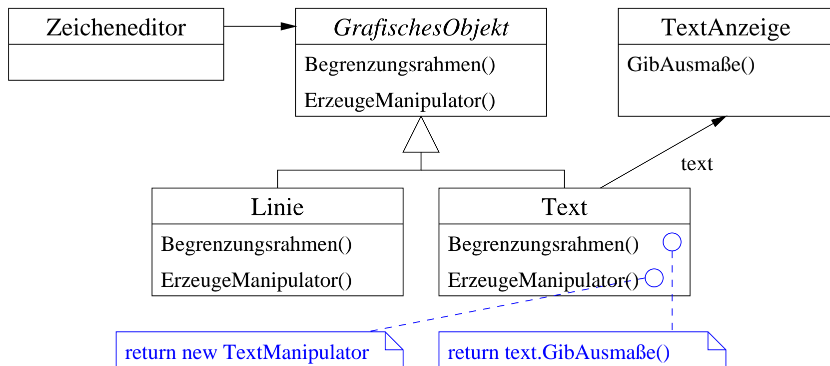
5.4.2.1 Adapter (auch: Wrapper)

- klassen- oder objektbasiert

Ziel

- Anpassung von Schnittstellen, z.B. bei Bibliotheksklassen
- hierzu ggf. Ergänzung fehlender Funktionalität

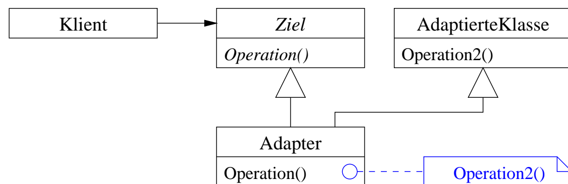
Adapter: Beispielanwendung



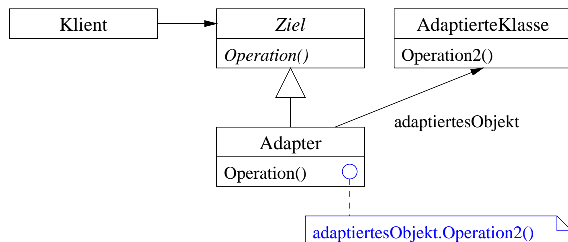
- die Adapterklasse `Text` passt die Klasse `TextAnzeige` an die Schnittstelle von `GrafischesObjekt` an
- ähnlich: Anpassung von Datenbankschnittstellen

Adapter: Allgemeine Struktur

1) Klassenadapter:



2) Objektadapter:



Eigenschaften

1) Klassenadapter
   - Adapter kann genau eine Klasse adaptieren, nicht aber ihre Unterklassen
   - Adapter kann Teile der adaptierten Klasse modifizieren (überschreiben)
   - nur ein Objekt
2) Objektadapter
   - Adapter kann mit anzupassender Klasse und allen ihren Unterklassen zusammenarbeiten
   - kein (direktes) Überschreiben von Teilen der adaptierten Klasse
   - zwei Objekte

5.4.2.2 Dekorierer (Decorator)

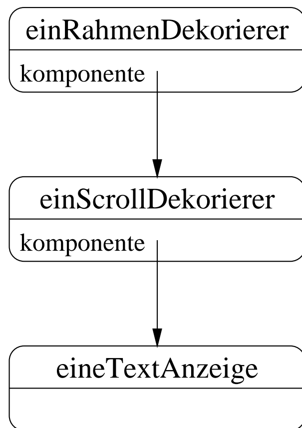
- objektbasiertes Strukturmuster

Ziel:

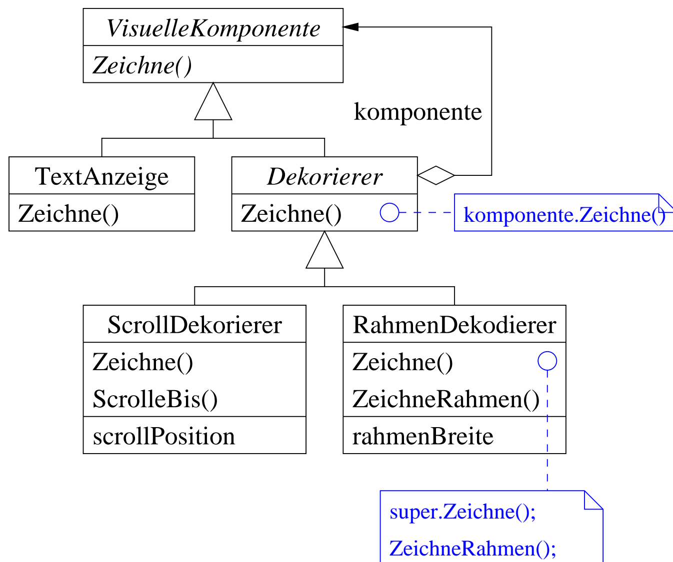
- erweitert Objekt dynamisch um Funktionalität
- flexiblere Alternative zur Unterklassenbildung

Dekorierer: Beispielanwendung

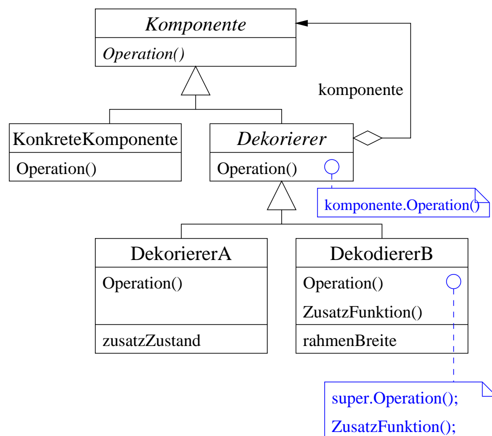
- grafisches Element auf Anforderung des Klienten umrahmen und/oder mit Scrollbalken versehen
- Dekoriererobjekt hat gleiche Schnittstelle wie dekorierte Objekte und leitet Anfragen weiter
- rekursives Dekorieren möglich



Dekorierer: Beispielanwendung (Fortsetzung)



Dekorierer: Allgemeine Struktur und Einordnung



Anwendbarkeit

- Objekte dynamisch und transparent um Funktionalität erweitern
- Zusatzfunktionalität entfernbar
- wenn Erweiterung durch Unterklassenbildung nicht sinnvoll, z.B. wegen Explosion der Klassenanzahl

Eigenschaften

- größere Flexibilität als statische Vererbung bzgl. Erweiterungsreihenfolge
- (hierarchisch hoch angesiedelte) Klassen werden nicht mit Funktionalität überlastet, sondern für einzelne Aspekte wird bei Bedarf Funktionalität durch weitere Klassen hinzugefügt
- Dekorierer und dekoriertes Objekt sind nicht identisch
  → Vorsicht bei Vergleich
- viele kleine Objekte
  → System ggf. schwerer zu verstehen und debuggen

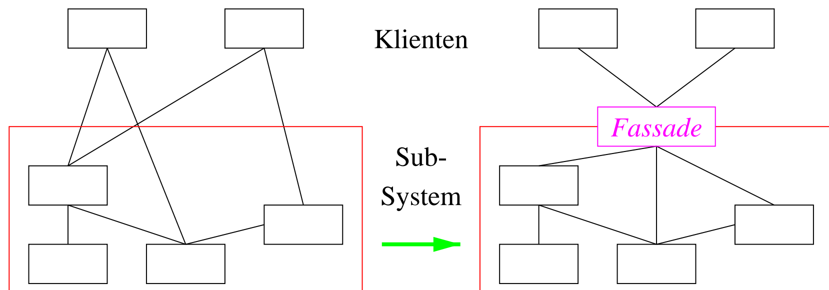
5.4.2.3 Fassade (Facade)

- objektbasiertes Strukturmuster

Ziel:

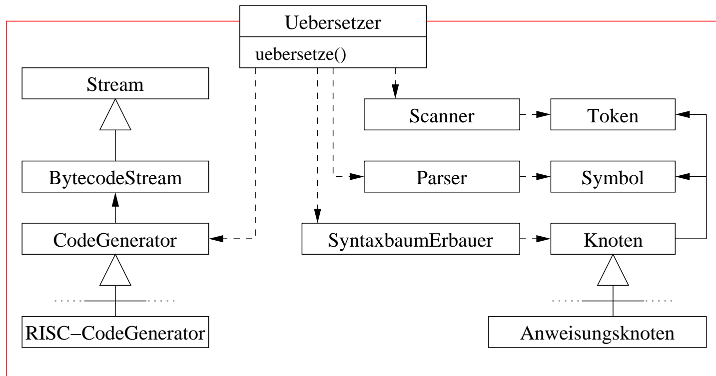
- Zusammenfassen der wichtigsten Schnittstellen eines Subsystems zu einheitlicher Schnittstelle
- Fassade bietet oft (auch) typische Kombinationen von Subsystemoperationen als eigene Operationen
- hierdurch einfachere Handhabung des Subsystems
- Durchgriff auf Einzelschnittstellen bleibt bei Bedarf möglich
- weniger Abhängigkeiten zwischen Subsystemen, insbesondere wenn Zugriff ausschließlich über Fassade (→ Schichtenarchitektur)

Fassade: Allgemeine Struktur



Fassade: Beispielanwendung

- Übersetzer-Subsystem enthält Klassen Scanner, Parser, Codegenerator usw. mit eigenen Schnittstellen
- Fassade kombiniert diese und bietet Operation `Uebersetze()`



Einordnung von Fassaden

Anwendbarkeit:

- wenn einfache Schnittstelle zu komplexem Subsystem hilfreich

Eigenschaften:

- Klienten kooperieren mit nur einem Fassadenobjekt statt mehreren Subsystemobjekten
- Änderungen innerhalb des Subsystems wirken sich nicht auf die Klienten aus, wenn Zugriffe nur über Fassade erfolgen
  → flexibler, leichter wartbar, leichter portierbar

5.4.2.4 Kompositum (Composite)

- objektbasiertes Strukturmuster

Ziel:

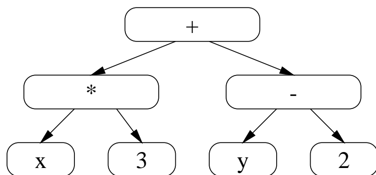
- Darstellung rekursiver Objektstrukturen
- Basiskomponenten und zusammengesetzte Komponenten einheitlich behandelbar

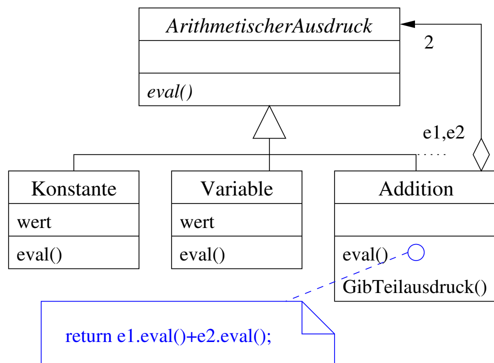
Kompositum: Beispielanwendungen

1) Komponenten eines Dokuments enthalten weitere Komponenten,
   z.B. Aufzählungsliste enthält Aufzählungsliste
2) Kontrollstrukturen einer Programmiersprache beliebig schachtelbar,
   z.B. Schleife enthält Schleife $\rightarrow$ Syntaxbaum
   Verarbeitung orientiert sich an Syntaxbaum (z.B. Typüberprüfung)

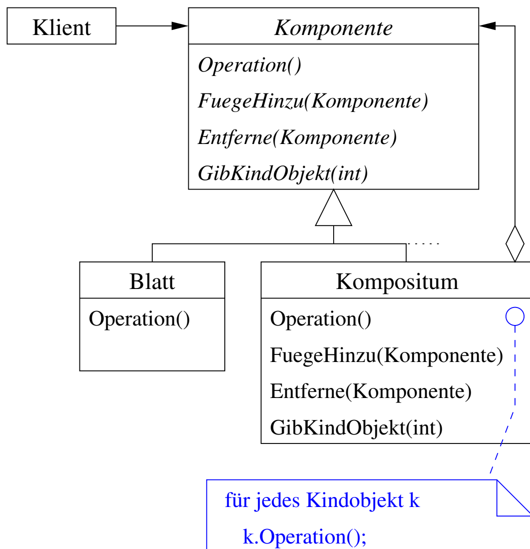
Kompositum: Beispielanwendungen (2)

3) ein arithmetischer Ausdruck besteht aus Teilausdrücken; die Auswertung orientiert sich an der entsprechenden Baumstruktur





Kompositum: Allgemeine Struktur



Einordnung des Kompositum-Musters

Anwendbarkeit:

- anwendbar bei rekursiven Objektstrukturen

Eigenschaften:

- Klientencode unabhängig von Art des vorliegenden Kompositums (elementar oder zusammengesetzt)
- neue Kompositions- und Blattklassen leicht ergänzbar

Verwandte Muster:

- wird oft zusammen mit Dekorierermuster verwendet
- ggf. Blätter als Fliegengewichte
- Iteratormuster zur Traversierung

5.4.2.5 Stellvertreter (Proxy, Surrogate)

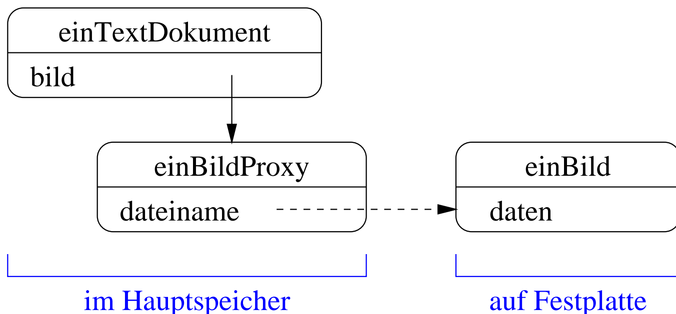
- objektbasiertes Strukturmuster

Ziel:

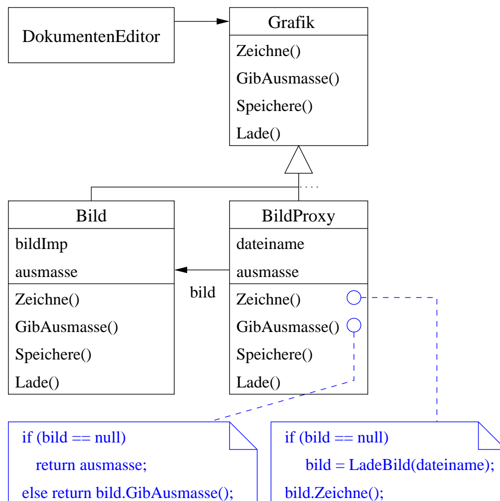
- kontrolliere Zugriff auf Objekt durch vorgelagerten Stellvertreter
- z.B. bei umfangreichen Objekten auf Festplatte
  (z.B. Bild, Audio, Video)

Stellvertreter: Beispielanwendung

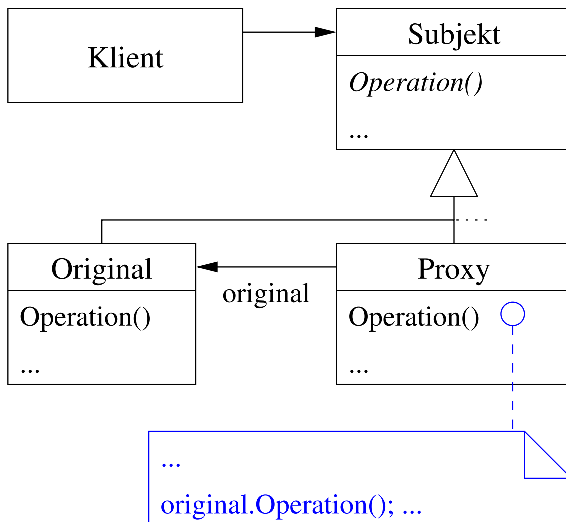
- ein grafischer Editor lädt Bilder erst bei Bedarf von der Festplatte
- bis dahin werden sie durch Stellvertreter repräsentiert
- der Stellvertreter veranlasst das rechtzeitige Laden des Bildobjektes



Stellvertreter: Beispielanwendung (Fortsetzung)



Stellvertreter: Allgemeine Struktur



Anwendbarkeit

1) Remote-Proxy:
   - zur Kommunikation mit Server-Objekt, z.B. bei CORBA
2) virtuelles Proxy: (vgl. Bsp.)
   - erzeugt teure Objekte erst bei Bedarf
   - verzögert Kosten für Erstellung und Initialisierung
3) Schutz-Proxy:
   - kontrolliert den Zugriff auf das Originalobjekt, z.B. Zugriffsrechte
4) Smart Reference:
   - ähnelt Zeiger, jedoch
   - führt Zusatzoperation aus
   - z.B. Reference-Counting, Sperren, Copy-on-Write

5.4.3 Verhaltensmuster

- weisen Objekten Zuständigkeiten zu
- regeln Interaktion zwischen Objekten

5.4.3.1 Befehl (Command)

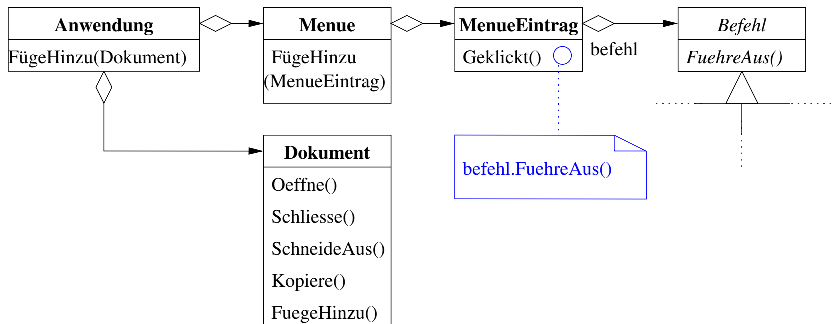
- objektbasiertes Verhaltensmuster

Ziel

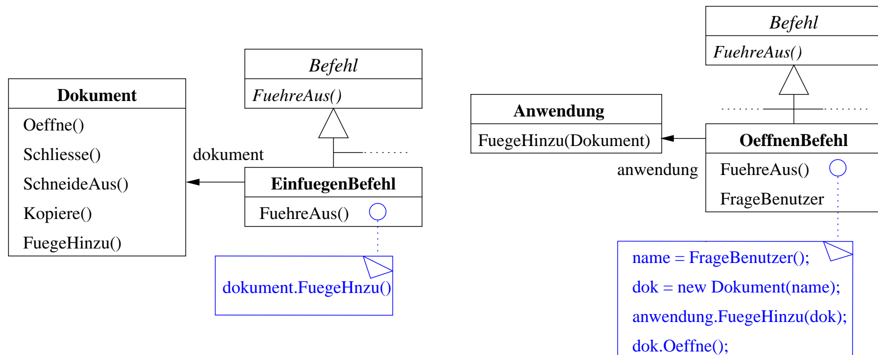
- kapselt Befehl als Objekt
- entkoppelt Auslöser und Bearbeiter einer Anfrage

Befehl: Beispielanwendung

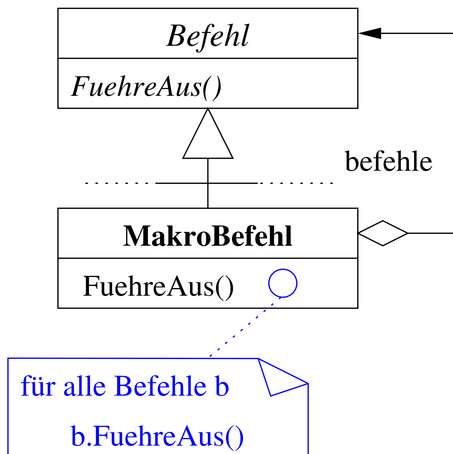
- Steuerungselemente einer Klassenbibliothek sollen Anfragen an unbekannte Anwendungsobjekte richten
- z.B. Knöpfe, Menüs, . . .
- hierzu: Anfrage als Objekt



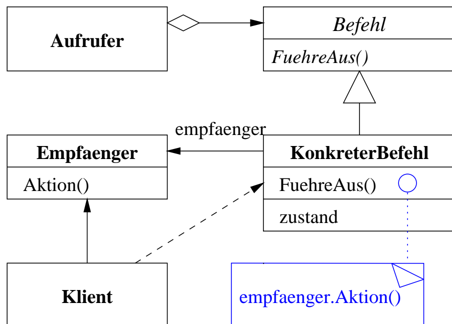
Befehl: Beispielanwendung (Fortsetzung)

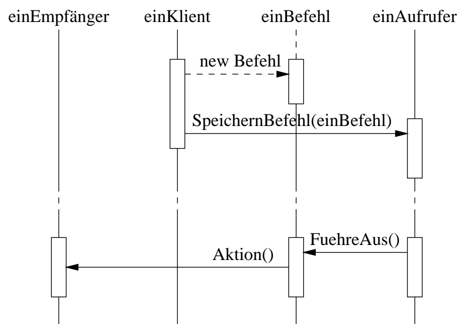


Befehlsmakros



Befehl: Allgemeine Struktur





Anwendbarkeit des Befehlsmusters

- Objekte mit einer Operation parametrisieren

  → OO-Ersatz für Callbacks
- Befehlsmakros
- Operationen eines Objekts austauschbar
- erlaubt Befehle als Parameter

  (vgl. Funktion höherer Ordnung, jedoch weniger typsicher)
- Befehlsobjekte in andere Adressräume übermitteln
- Manipulation von Befehlen auf “Meta-Ebene”, z.B.
  - Anfragen in Warteschlange verwalten
  - Logbuch führen
  - undo unterstützen
  - Änderungen protokollieren, Recovery → Transaktionen

5.4.3.2 Beobachter (Observer)

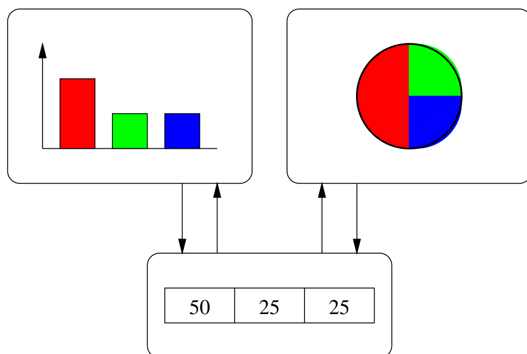
- objektbasiertes Verhaltensmuster

Ziel

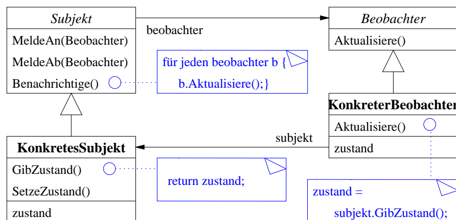
- $1 : n$-Abhängigkeit zwischen Objekten pflegen
- bei Änderung des einen Objekts werden alle abhängigen Objekte informiert und automatisch aktualisiert

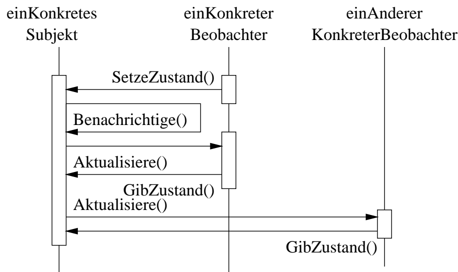
Beobachter: Beispielanwendung

- Konsistenz zwischen Tabelle und Diagrammen aufrecht erhalten



Beobachter: Allgemeine Struktur





Einordnung des Beobachtermusters

Anwendbarkeit:

- Objekte mit $\geq$ 2 voneinander abhängigen Aspekten

Eigenschaften:

- Subjekt und Beobachter können voneinander unabhängig geändert werden, denn
- Subjekt kennt von Beobachter nur OID
- Broadcasting
- neue Beobachter leicht ergänzbar
- das simple Protokoll kann bei minimalen Zustandänderungen zu aufwendigen Operationen beim Beobachter führen, da er nicht weiß, welches Detail geändert wurde $\rightarrow$ verfeinertes Protokoll

5.4.3.3 Besucher (Visitor)

- objektbasiertes Verhaltensmuster

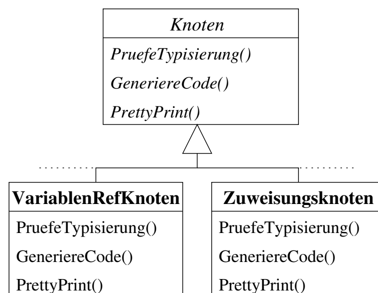
Ziel

- kapselt eine auf den Komponenten einer Objektstruktur auszuführende Operation als Objekt
- erlaubt neue Operationen ohne Klassen der Objektstrukur zu erweitern

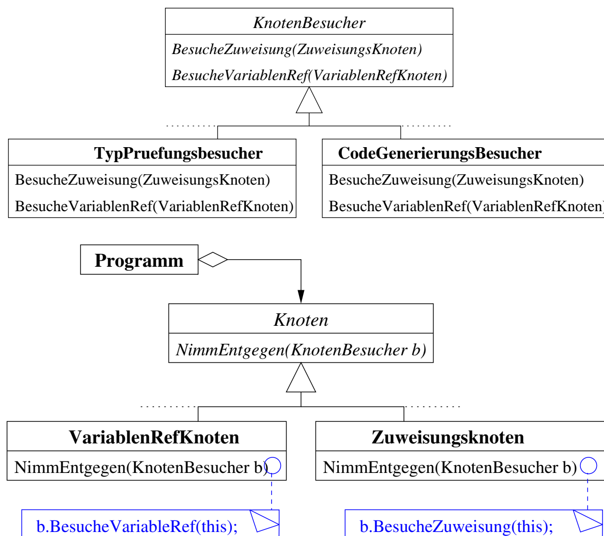
Besucher: Beispielanwendung

- ein Syntaxbaum wird auf verschiedene Weisen bearbeitet:
  Typüberprüfung, Codegenerierung, Pretty Printing, ...
- naiv: alle zugehörigen Operationen in Knotenklassen
  → schwer verständlich, komplex, schwer änderbar
- jede Bearbeitungsphase durch Besucher realisieren

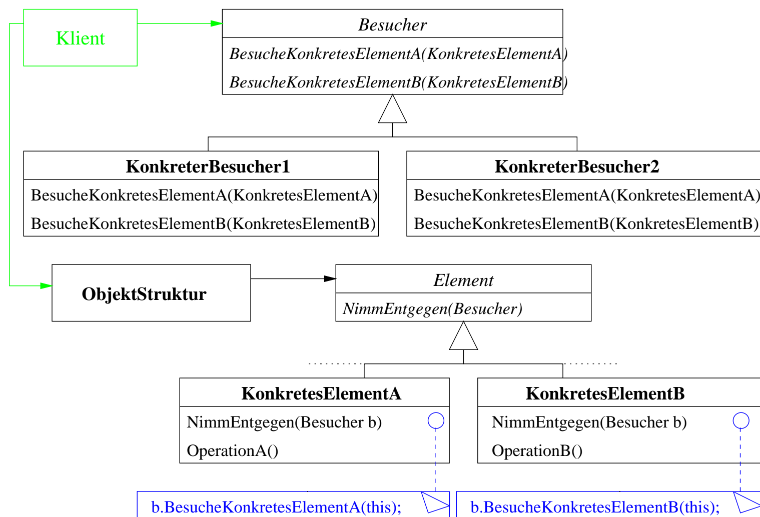
ohne Besucher:



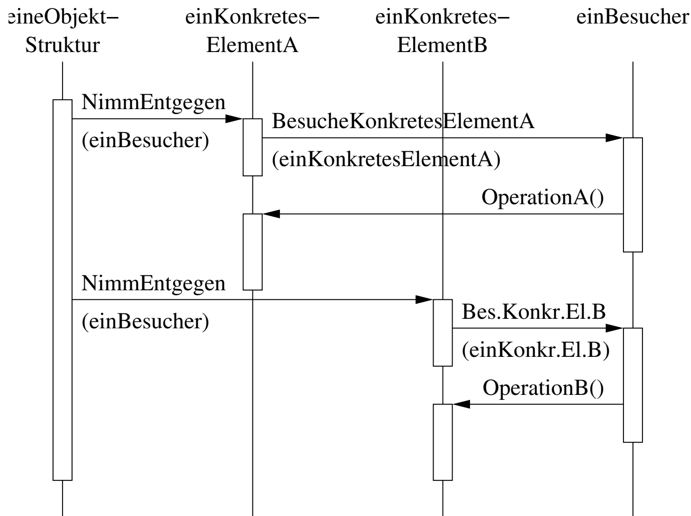
Syntaxbaum mit Besuchern



Besucher: Allgemeine Struktur



Besucher: Sequenzdiagramm



Einordnung des Besuchermusters

Anwendbarkeit:

- bei Objektstruktur mit mehreren voneinander unabhängigen Schnittstellen
- wenn für eine stabile Objektstruktur häufig neue Operationen erstellt werden
- nicht geeignet bei instabiler Objektstruktur (aufwändige Anpassung!)

Eigenschaften:

- erleichtert das Hinzufügen von Operationen zu komplexen Objektstrukturen
- erleichtert Verständnis und Wartung durch Extraktion und Zusammenfassung verwandter Operationen
- Besucher können Zustandsinformationen ansammeln
- Besucher benötigt Get- und Set-Methoden, ggf. Verschlechterung der Kapselung

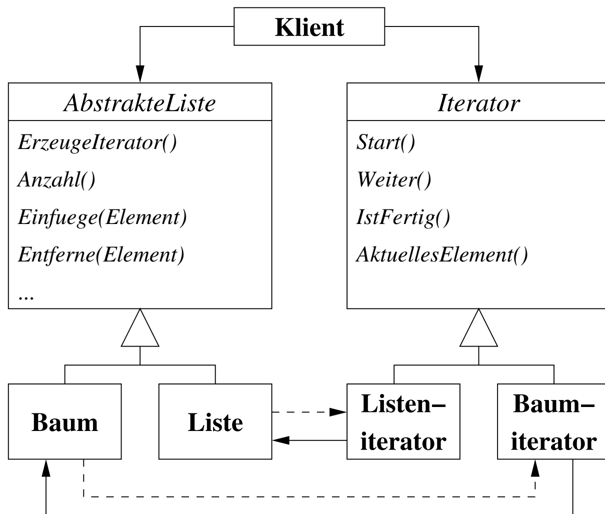
5.4.3.4 Iterator (Cursor)

- objektbasiertes Verhaltensmuster
- vgl. Java-Interface `Iterator`

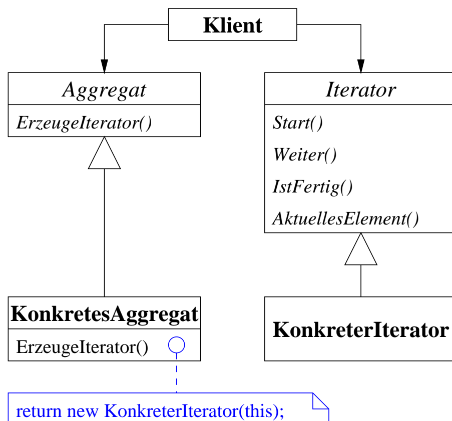
Ziel

- sequentieller Zugriff auf Komponenten eines zusammengesetzten Objekts
- Repräsentation des Objekts bleibt versteckt

Iterator: Beispielanwendung



Iterator: Allgemeine Struktur



- `KonkreterIterator` verwaltet das aktuelle Objekt und kann das nachfolgende Objekt ermitteln

Einordnung des Iterator-Musters

Anwendbarkeit:

- ermöglicht gleichzeitig mehrere Traversierungen
- ermöglicht unterschiedliche Traversierungsarten
- erlaubt “polymorphe Iteration”,
  d.h. Klient bleibt unabh. von konkreter Objektstruktur

Eigenschaften:

- Traversierungsart leicht änderbar
- vereinfacht Kollektionsklassen
  (keine eigene Iterationsschnittstelle mehr)

5.4.3.5 Strategie (Strategy, Policy)

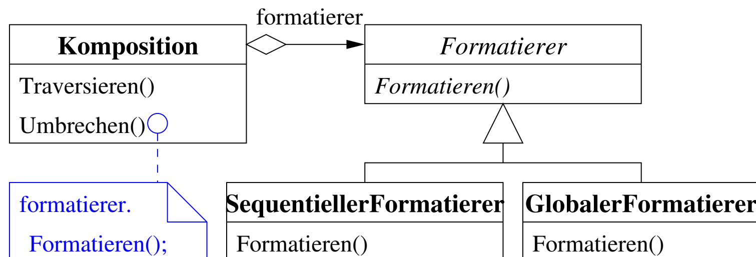
- objektbasiertes Verhaltensmuster

Ziel:

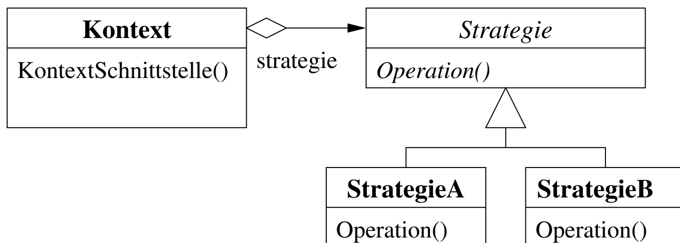
- kapselt Algorithmen
- vereinfacht ihren Austausch

Strategie: Beispielanwendung

- Text in Zeilen umbrechen
- hierfür gibt es viele Algorithmen
- nicht sinnvoll: Algorithmen in Klientenklassen integrieren
- denn dann: schlecht austauschbar; daher: Algorithmen kapseln



Strategie: Allgemeine Struktur



- anwendbar bei Familien von Algorithmen mit gleicher Schnittstelle

5.5 Kombination von Entwurfsmustern

- Entwufsmuster werden häufig kombiniert

Beispiel: MVC (Model View Controller)

- kombiniert u.a. Beobachter und Strategie
- Model: Kern der Anwendung; Daten und zugehörige Methoden
- View: zeigt Model-Zustand an
- Controller: übersetzt Ereignisfolge in Folge von Dienstleistungen des Models

MVC: Klassen- und Sequenz-Diagramme