

Softwaretechnik 1 - 4. Tutorium

Tutorium 03

Felix Bachmann | 26.06.2017

KIT - INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION (IPD)



- 1 Orga
- 2 Recap
- 3 Stellvertreter
- 4 Vermittler
- 5 Gruppenarbeit
- 6 Memento
- 7 Befehl
- 8 Tipps

Ansage der Übungsleiter

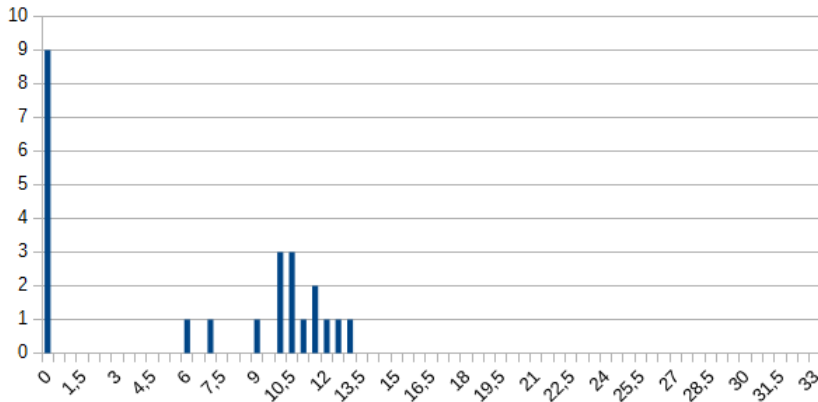
- ab jetzt keine Abgabe per Mail mehr!
⇒ auch nicht in Ausnahmefällen

Ansage der Übungsleiter

- ab jetzt keine Abgabe per Mail mehr!
⇒ auch nicht in Ausnahmefällen
- keine semi-handschriftlichen Abgaben mehr!

4. Übungsblatt Statistik

n=24



Ø 6,45 bzw. 10,33 von (bisher) 15+1

Allgemein

- Form bei handschriftlichen Abgaben. . .

Aufgabe 1 (Zustandsdiagramm - LEZ): \emptyset 2,81 bzw. 4,5 von 5+1

Aufgabe 1 (Zustandsdiagramm - LEZ): \emptyset 2,81 bzw. 4,5 von 5+1

- Hierarchie sinnvoll, wenn aus mehreren Zuständen gleiche Übergänge in den gleichen Zustand gehen

Aufgabe 1 (Zustandsdiagramm - LEZ): \emptyset 2,81 bzw. 4,5 von 5+1

- Hierarchie sinnvoll, wenn aus mehreren Zuständen gleiche Übergänge in den gleichen Zustand gehen
- nach VL gibt es im Zustandsdiagramm kein “Karo”

Aufgabe 1 (Zustandsdiagramm - LEZ): Ø 2,81 bzw. 4,5 von 5+1

- Hierarchie sinnvoll, wenn aus mehreren Zuständen gleiche Übergänge in den gleichen Zustand gehen
- nach VL gibt es im Zustandsdiagramm kein “Karo”
- “Versehen Sie die Zustandsübergänge mit Ereignissen **und** Operationen.”

Aufgabe 1 (Zustandsdiagramm - LEZ): \emptyset 2,81 bzw. 4,5 von 5+1

- Hierarchie sinnvoll, wenn aus mehreren Zuständen gleiche Übergänge in den gleichen Zustand gehen
- nach VL gibt es im Zustandsdiagramm kein “Karo”
- “Versehen Sie die Zustandsübergänge mit Ereignissen **und** Operationen.”
⇒ kann in Klausur bei Nichtbeachtung Punktabzug geben

Aufgabe 2 (Abbottsche Methode): Ø 1,73 bzw. 3,19 von 5

Aufgabe 2 (Abbottsche Methode): Ø 1,73 bzw. 3,19 von 5

- bei “auseinandergezogenen Verben“ alle Teile des Verbs markieren

Aufgabe 2 (Abbottsche Methode): Ø 1,73 bzw. 3,19 von 5

- bei “auseinandergezogenen Verben” alle Teile des Verbs markieren
z.B “teilnehmen” \Rightarrow “Studenten nehmen an VL teil”

Aufgabe 2 (Abbottsche Methode): Ø 1,73 bzw. 3,19 von 5

- bei “auseinandergezogenen Verben” alle Teile des Verbs markieren
z.B “teilnehmen” \Rightarrow “Studenten nehmen an VL teil”
- Worte kommen mehrfach vor \Rightarrow jedes Mal markieren!

Aufgabe 2 (Abbottsche Methode): Ø 1,73 bzw. 3,19 von 5

- bei “auseinandergezogenen Verben“ alle Teile des Verbs markieren
z.B “teilnehmen“ \implies “Studenten nehmen an VL teil“
- Worte kommen mehrfach vor \implies jedes Mal markieren!
- bei jedem “ist“, “sind“, etc. Vererbung

Aufgabe 2 (Abbottsche Methode): Ø 1,73 bzw. 3,19 von 5

- bei “auseinandergezogenen Verben“ alle Teile des Verbs markieren
z.B “teilnehmen“ \implies “Studenten nehmen an VL teil“
- Worte kommen mehrfach vor \implies jedes Mal markieren!
- bei jedem “ist“, “sind“, etc. Vererbung
- “wissenschaftlicher Mitarbeiter“ = Attribut und Klasse

Aufgabe 2 (Abbottsche Methode): Ø 1,73 bzw. 3,19 von 5

- bei “auseinandergezogenen Verben“ alle Teile des Verbs markieren
z.B “teilnehmen“ \Rightarrow “Studenten nehmen an VL teil“
- Worte kommen mehrfach vor \Rightarrow jedes Mal markieren!
- bei jedem “ist“, “sind“, etc. Vererbung
- “wissenschaftlicher Mitarbeiter“ = Attribut und Klasse

Aufgabe 3 (iMage-GUI): Ø (tbd)

- (nächstes Mal)

Aufgabe 4 (Geheimnisprinzip): \emptyset 1,92 bzw. 3,07 von 5

Aufgabe 4 (Geheimnisprinzip): \emptyset 1,92 bzw. 3,07 von 5

- nicht nur die öffentlichen Konstanten sind problematisch, sondern auch die getter und setter

Aufgabe 4 (Geheimnisprinzip): Ø 1,92 bzw. 3,07 von 5

- nicht nur die öffentlichen Konstanten sind problematisch, sondern auch die getter und setter
⇒ die Entscheidung den Zustand intern als int zu repräsentieren muss versteckt werden

Aufgabe 4 (Geheimnisprinzip): \emptyset 1,92 bzw. 3,07 von 5

- nicht nur die öffentlichen Konstanten sind problematisch, sondern auch die getter und setter
 - \implies die Entscheidung den Zustand intern als int zu repräsentieren muss versteckt werden
 - \implies nach außen immer boolean benutzen (wohldefiniert!)

Was bisher geschah..

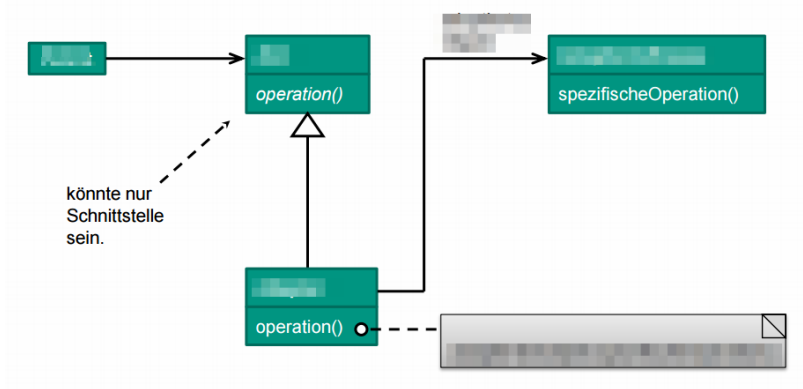
- haben uns erste Entkopplungsmuster angeschaut

Was bisher geschah..

- haben uns erste Entkopplungsmuster angeschaut
⇒ Beobachter, Iterator, Adapter

Was bisher geschah..

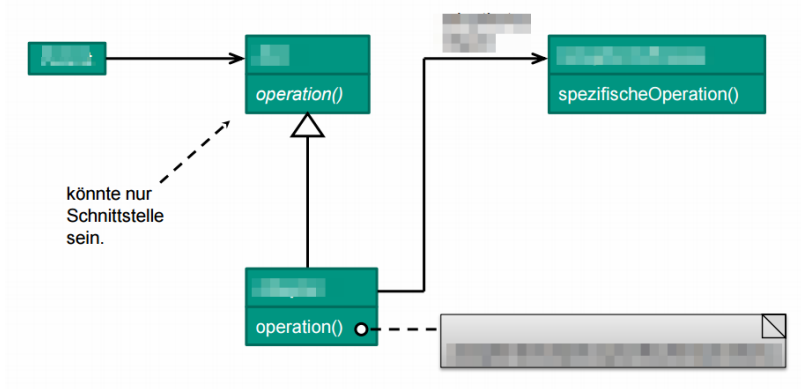
- haben uns erste Entkopplungsmuster angeschaut
⇒ Beobachter, Iterator, Adapter



Welches Entwurfsmuster?

Was bisher geschah..

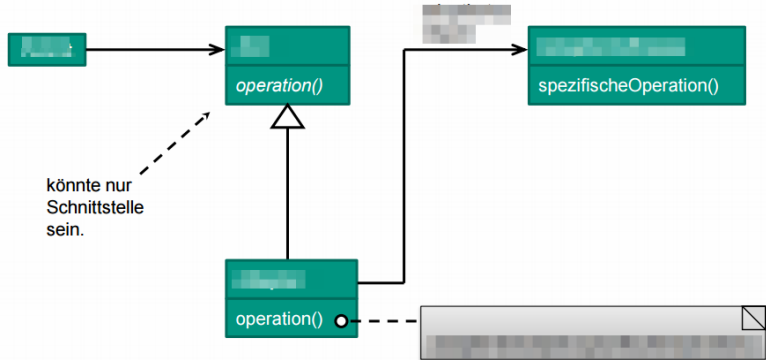
- haben uns erste Entkopplungsmuster angeschaut
⇒ Beobachter, Iterator, Adapter



Welches Entwurfsmuster? (Objekt-)Adapter

Was bisher geschah..

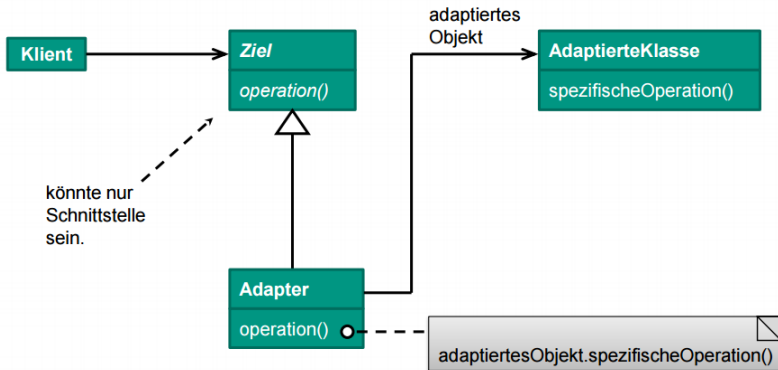
- haben uns erste Entkopplungsmuster angeschaut
⇒ Beobachter, Iterator, Adapter



Welche Klassen?

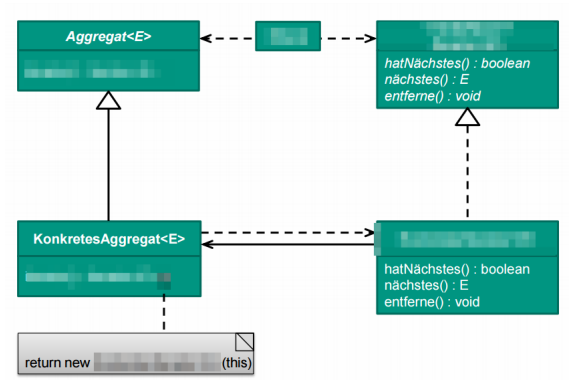
Was bisher geschah..

- haben uns erste Entkopplungsmuster angeschaut
 - ⇒ Beobachter, Iterator, Adapter



Was bisher geschah..

- haben uns erste Entkopplungsmuster angeschaut
⇒ Beobachter, Iterator, Adapter



Welches Entwurfsmuster?

Orga

○○○○○○

Recap

○○○●○○○○

Stellvertreter

○○

Vermittler

○○

Gruppenarbeit

○○○○○○○○○○

Memento

○○○○

Befehl

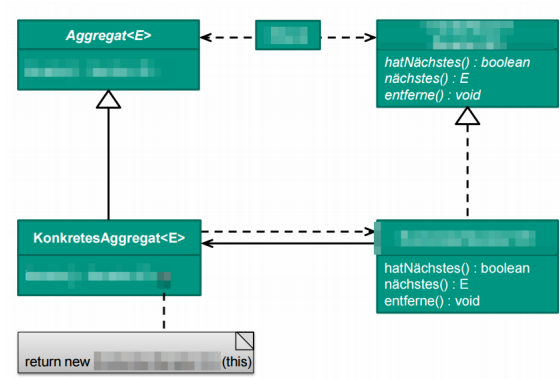
○○○○○○

Tipps

○○○○○

Was bisher geschah..

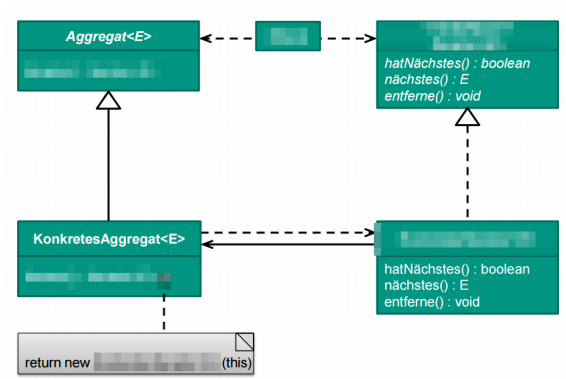
- haben uns erste Entkopplungsmuster angeschaut
⇒ Beobachter, Iterator, Adapter



Welches Entwurfsmuster? Iterator

Was bisher geschah..

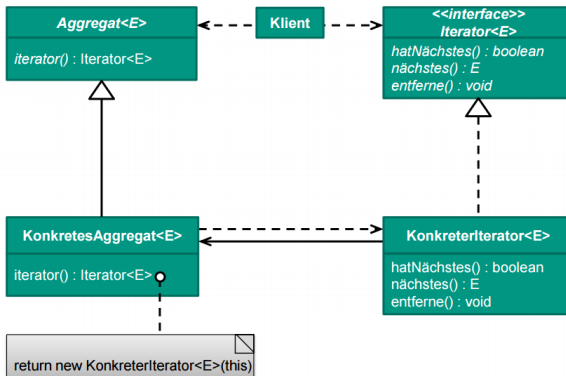
- haben uns erste Entkopplungsmuster angeschaut
⇒ Beobachter, Iterator, Adapter



Welche Klassen und Methoden?

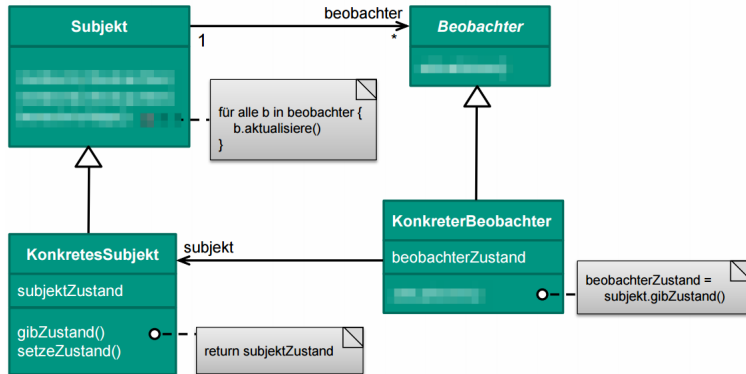
Was bisher geschah..

- haben uns erste Entkopplungsmuster angeschaut
⇒ Beobachter, Iterator, Adapter



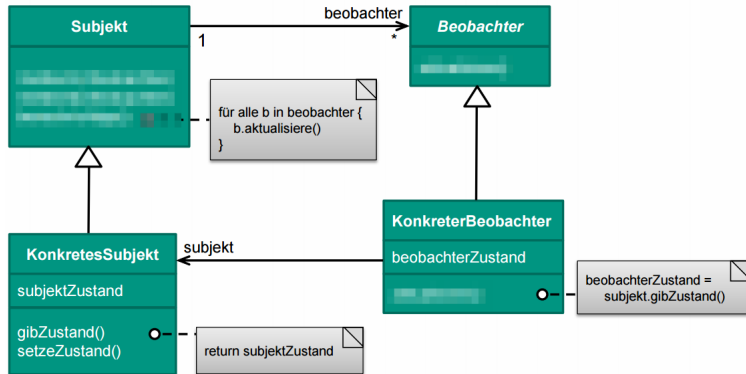
Was bisher geschah..

- haben uns erste Entkopplungsmuster angeschaut
⇒ Beobachter, Iterator, Adapter



Was bisher geschah..

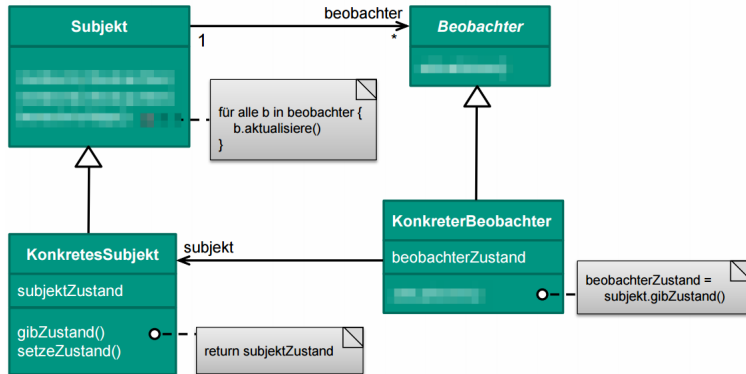
- haben uns erste Entkopplungsmuster angeschaut
⇒ Beobachter, Iterator, Adapter



Ist wohl ein Beobachter :)

Was bisher geschah..

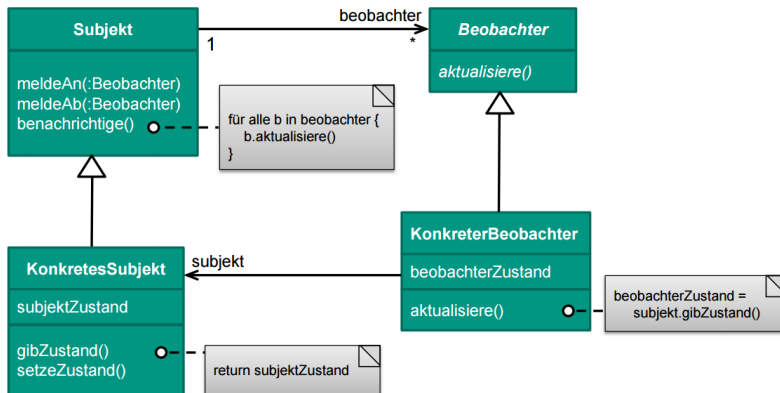
- haben uns erste Entkopplungsmuster angeschaut
⇒ Beobachter, Iterator, Adapter



Ist wohl ein Beobachter :) Methoden?

Was bisher geschah..

- haben uns erste Entkopplungsmuster angeschaut
⇒ Beobachter, Iterator, Adapter



■ Entkopplungs-Muster

- Adapter **fertig**
- Beobachter **fertig**
- Iterator **fertig**
- **Stellvertreter**
- **Vermittler**
- (Brücke)

■ Varianten-Muster

■ Zustandshandhabungs-Muster

■ Steuerungs-Muster

■ Bequemlichkeits-Muster

Problem

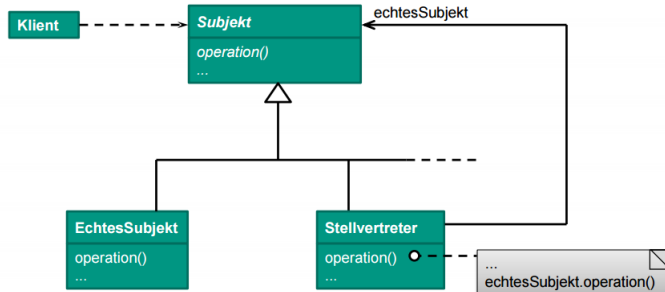
- wollen Zugriff auf ein Objekt kontrollieren, ohne seine Klasse zu ändern

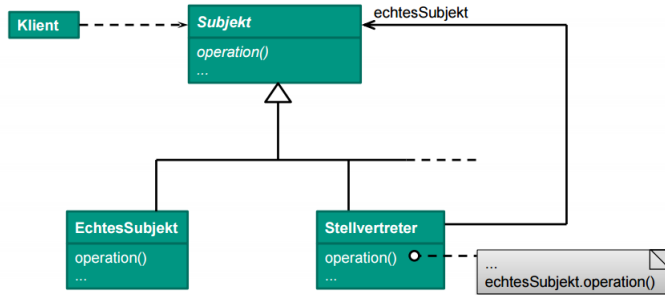
Problem

- wollen Zugriff auf ein Objekt kontrollieren, ohne seine Klasse zu ändern
⇒ Stellvertreter macht Zugriffskontrolle

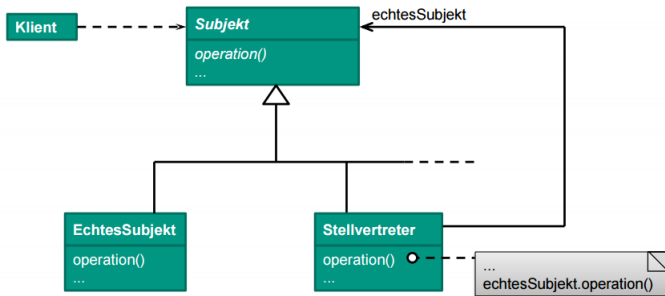
Problem

- wollen Zugriff auf ein Objekt kontrollieren, ohne seine Klasse zu ändern
⇒ Stellvertreter macht Zugriffskontrolle



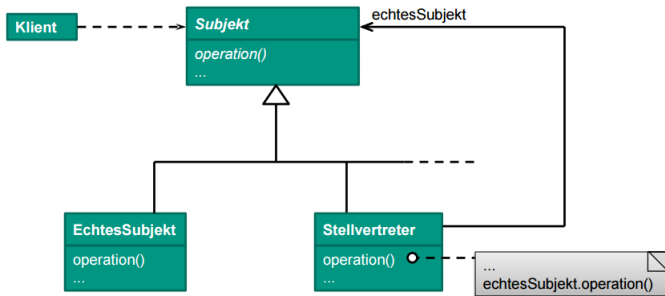


Entkopplung?



Entkopplung?

- Klient hat keinen direkten Zugriff auf das echte Subjekt



Entkopplung?

- Klient hat keinen direkten Zugriff auf das echte Subjekt
- Stellvertreter hat Relation zu Oberklasse (!), echtes Subjekt austauschbar

Problem

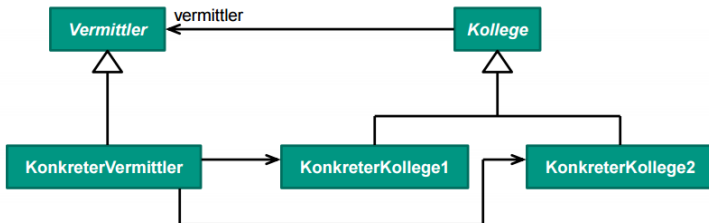
- mehrere voneinander abhängige Objekte

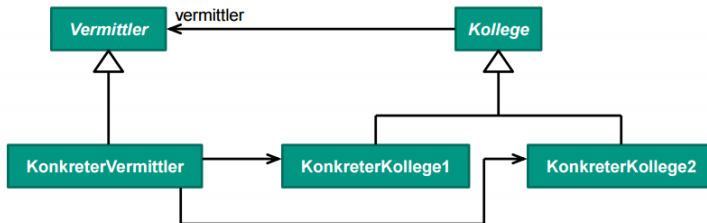
Problem

- mehrere voneinander abhängige Objekte
⇒ Zustände der Objekte von anderen Zuständen abhängig

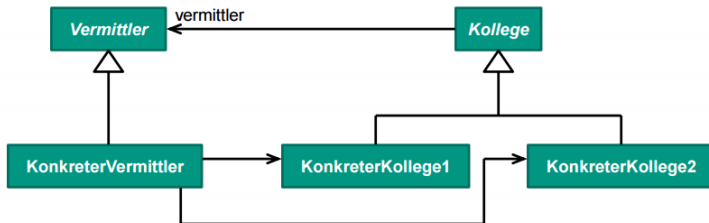
Problem

- mehrere voneinander abhängige Objekte
⇒ Zustände der Objekte von anderen Zuständen abhängig



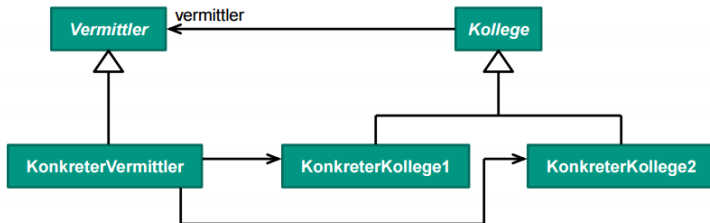


Entkopplung?



Entkopplung?

- Kollegen kennen sich nicht direkt



Entkopplung?

- Kollegen kennen sich nicht direkt
⇒ Hinzufügen eines Kollegen erfordert keine Änderung der alten Kollegen

- Entkopplungs-Muster **fertig**
- **Varianten-Muster**
 - (Abstrakte Fabrik)
 - (Besucher)
 - **Schablonenmethode**
 - **Fabrikmethode**
 - **Kompositum**
 - Strategie **fertig**
 - **Dekorierer**
- Zustandshandhabungs-Muster
- Steuerungs-Muster
- Bequemlichkeits-Muster

Übergeordnetes Ziel

- Gemeinsamkeiten herausziehen und an einer Stelle beschreiben

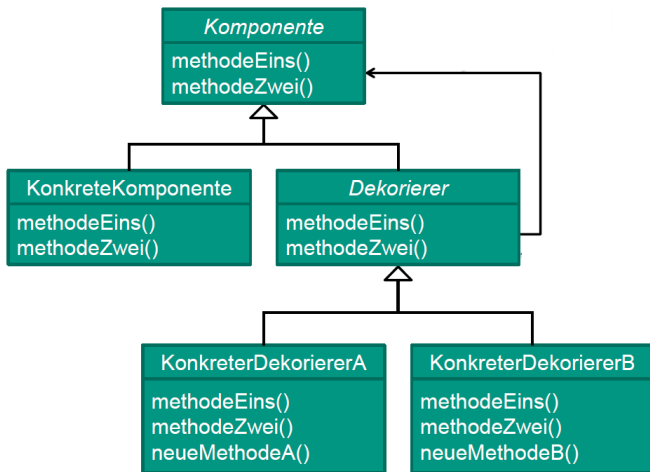
Übergeordnetes Ziel

- Gemeinsamkeiten herausziehen und an einer Stelle beschreiben
⇒ keine Wiederholung desselben Codes

Übergeordnetes Ziel

- Gemeinsamkeiten herausziehen und an einer Stelle beschreiben
 - ⇒ keine Wiederholung desselben Codes
 - ⇒ bessere Wartbarkeit/Erweiterbarkeit

- 1 ihr kriegt pro Reihe eine Aufgabe
- 2 ihr habt Zeit zum Bearbeiten
- 3 Abgleichung mit Musterlösung
- 4 ihr stellt den anderen eure Lösung vor



Wo Gemeinsamkeiten?

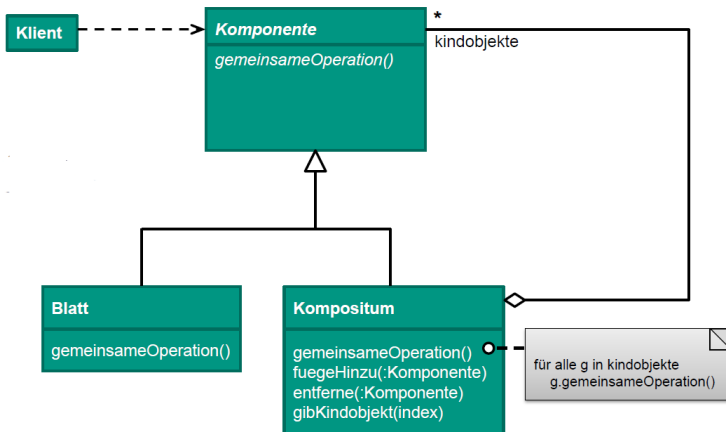
Die beiden Methoden `methodeEins()` und `methodeZwei()`.

Wo Variation?

In den KonkretenDekorierern bzw. ihren Methoden. Hier: `neueMethodeA()`, `neueMethodeB()`.

Wozu Instanzvariable?

Weiterleitung von Aufrufen der `methodeEins()` und `methodeZwei()` an die KonkreteKomponente.



Wo Gemeinsamkeiten?

gemeinsameOperation().

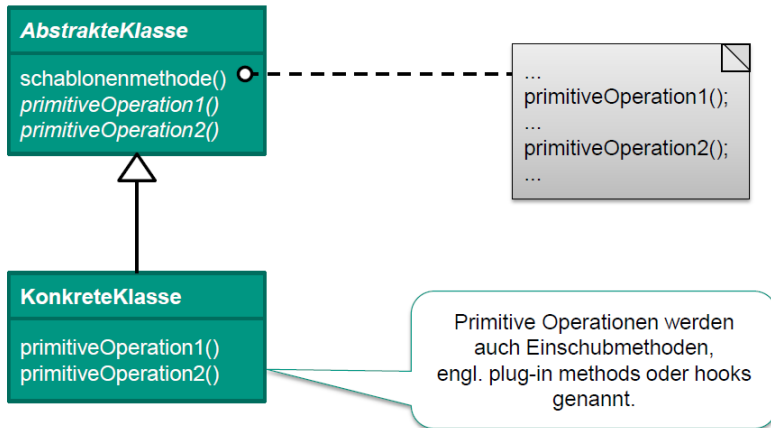
Wo Variation?

In Blatt/Kompositum-Klassen mit verschiedenen zusätzlichen Operationen.

Zusammengesetzt vs. nicht-zusammengesetzt

Kompositum = zusammengesetzt, Blatt = nicht-zusammengesetzt

Vorstellung Schablonenmethode



Wo Gemeinsamkeiten?

Reihenfolge der Methodenaufrufe in der Schablonenmethode.

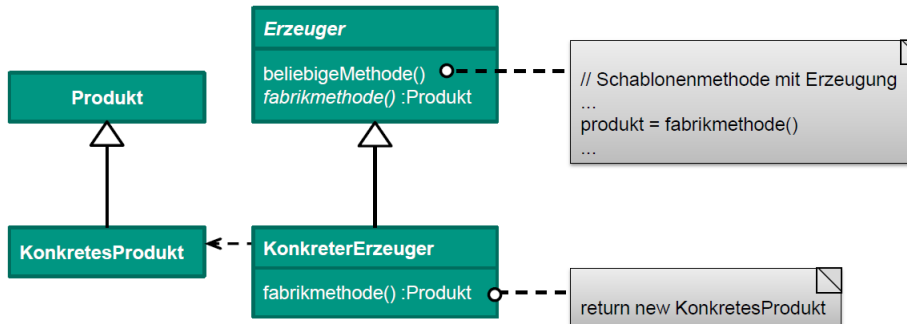
Wo Variation?

In den Einschubmethoden. (hier: `primitiveOperation1()` und `primitiveOperation2()`)

Schablonenmethode vs. Einschubmethode

Einschubmethode ist eine der Methoden, die von der Schablonenmethode aufgerufen wird und deren Implementierung in den Unterklassen stattfindet.

Vorstellung Fabrikmethode



Wo Gemeinsamkeiten?

Reihenfolge der Methodenaufrufe in der beliebigenMethode().

Wo Variation?

In der Fabrikmethode.

Klasse des Objekts, Oberklasse, Unterklasse

Klasse des Objekts = KonkretesProdukt, Oberklasse = Produkt,
Unterklasse = KonkreterErzeuger

Unterschied zu Schablonenmethode?

Fabrikmethode benutzen, wenn ein Objekt erzeugt wird. Fabrikmethode ist Einschubmethode des Musters "Schablonenmethode".

Wahr/falsch

Fabrikmethode ist eine Einschubmethode, keine Schablonenmethode.

- Entkopplungs-Muster fertig
- Varianten-Muster fertig
- **Zustandshandhabungs-Muster**
 - (Einzelstück)
 - (Fliegengewicht)
 - **Memento**
 - (Prototyp)
 - (Zustand)
- Steuerungs-Muster
- Bequemlichkeits-Muster

Übergeordnetes Ziel

- den Zustand eines Objektes beschreiben (wer hätt's gedacht? :D)

Übergeordnetes Ziel

- den Zustand eines Objektes beschreiben (wer hätt's gedacht? :D)
- aber unabhängig von dem Zweck des Objekts!

Problem

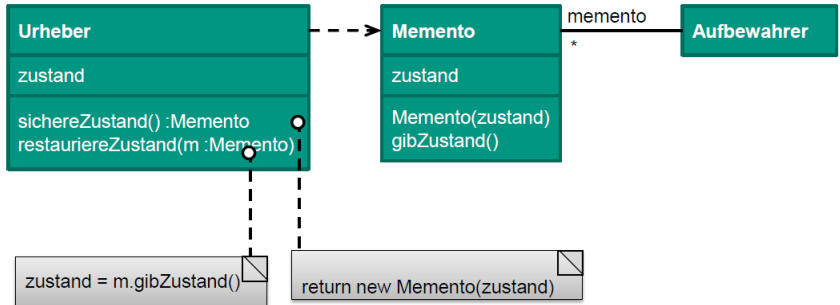
- internen Zustand eines Objekts “externalisieren“, um z.B. Zurücksetzen möglich zu machen

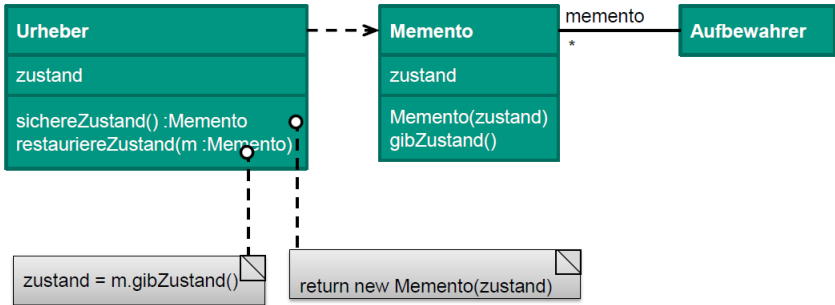
Problem

- internen Zustand eines Objekts “externalisieren“, um z.B. Zurücksetzen möglich zu machen
- ohne Kapselung zu verletzen!

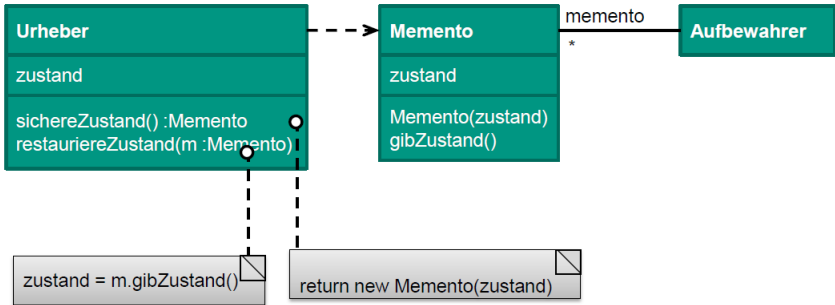
Problem

- internen Zustand eines Objekts “externalisieren“, um z.B. Zurücksetzen möglich zu machen
- ohne Kapselung zu verletzen!



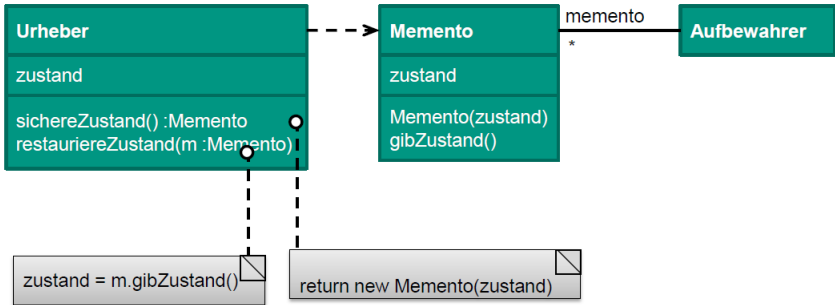


Problem gelöst?



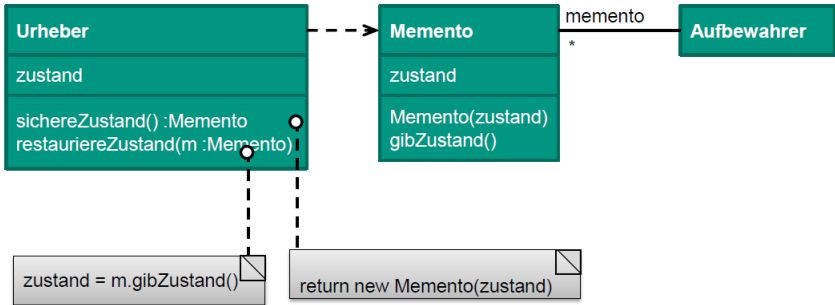
Problem gelöst?

- Ja



Problem gelöst?

- Ja
 - Zustand durch Memento externalisiert



Problem gelöst?

- Ja
 - Zustand durch Memento externalisiert
 - Kapselung nicht verletzt (Nutzer ruft nur `sichereZustand()` auf und kriegt neuen Memento)

- Entkopplungs-Muster fertig
- Varianten-Muster fertig
- Zustandshandhabungs-Muster fertig
- **Steuerungs-Muster**
 - Befehl
 - (master/worker)
- Bequemlichkeits-Muster

Übergeordnetes Ziel

- steuern den Kontrollfluss

Übergeordnetes Ziel

- steuern den Kontrollfluss
⇒ zur richtigen Zeit richtige Methoden aufrufen

Problem

- Parametrisieren von Objekten mit einer auszuführenden Aktion

Problem

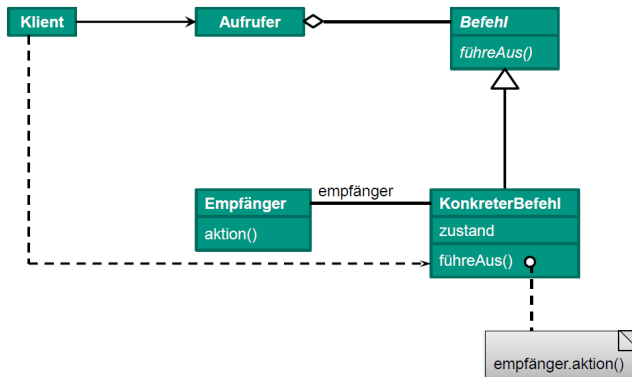
- Parametrisieren von Objekten mit einer auszuführenden Aktion
- komplexe Operationen aus primitiven Operationen aufbauen

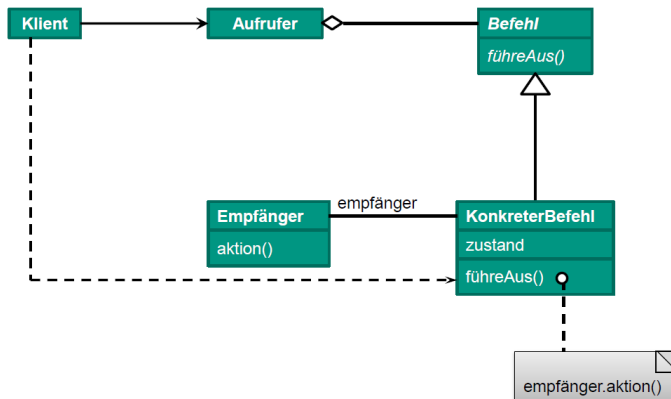
Problem

- Parametrisieren von Objekten mit einer auszuführenden Aktion
- komplexe Operationen aus primitiven Operationen aufbauen
⇒ Befehl nicht als Methode, sondern als Objekt modellieren

Problem

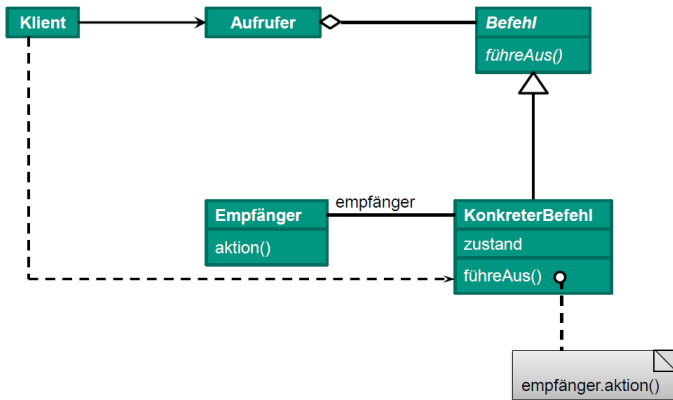
- Parametrisieren von Objekten mit einer auszuführenden Aktion
- komplexe Operationen aus primitiven Operationen aufbauen
 \Rightarrow Befehl nicht als Methode, sondern als Objekt modellieren





Was haben wir erreicht?

- Austauschbarkeit: Befehle unabhängig vom Aufrufer, universell einsetzbar



Was haben wir erreicht?

- Austauschbarkeit: Befehle unabhängig vom Aufrufer, universell einsetzbar
- Logik komplett im Befehl \Rightarrow keine Inkonsistenzen bei Benutzung

Wahr oder falsch?

- Bei dem Entwurfsmuster Befehl kennt der Empfänger den Befehl nicht, jedoch der Befehl den Empfänger.

Wahr oder falsch?

- Bei dem Entwurfsmuster Befehl kennt der Empfänger den Befehl nicht, jedoch der Befehl den Empfänger. **wahr**

Wahr oder falsch?

- Bei dem Entwurfsmuster Befehl kennt der Empfänger den Befehl nicht, jedoch der Befehl den Empfänger. **wahr**
- Ein Aufbewahrer im Entwurfsmuster Memento kann beliebig viele Mementos verwalten. Für die Restauration im Falle eines Reset ist er allerdings nicht verantwortlich.

Wahr oder falsch?

- Bei dem Entwurfsmuster Befehl kennt der Empfänger den Befehl nicht, jedoch der Befehl den Empfänger. **wahr**
- Ein Aufbewahrer im Entwurfsmuster Memento kann beliebig viele Mementos verwalten. Für die Restauration im Falle eines Reset ist er allerdings nicht verantwortlich. **wahr**

Wahr oder falsch?

- Bei dem Entwurfsmuster Befehl kennt der Empfänger den Befehl nicht, jedoch der Befehl den Empfänger. **wahr**
- Ein Aufbewahrer im Entwurfsmuster Memento kann beliebig viele Mementos verwalten. Für die Restauration im Falle eines Reset ist er allerdings nicht verantwortlich. **wahr**
- Die Fabrikmethode sorgt dafür, dass nur eine einzige Instanz einer Klasse fabriziert wird.

Wahr oder falsch?

- Bei dem Entwurfsmuster Befehl kennt der Empfänger den Befehl nicht, jedoch der Befehl den Empfänger. **wahr**
- Ein Aufbewahrer im Entwurfsmuster Memento kann beliebig viele Mementos verwalten. Für die Restauration im Falle eines Reset ist er allerdings nicht verantwortlich. **wahr**
- Die Fabrikmethode sorgt dafür, dass nur eine einzige Instanz einer Klasse fabriziert wird. **falsch**

Wahr oder falsch?

- Bei dem Entwurfsmuster Befehl kennt der Empfänger den Befehl nicht, jedoch der Befehl den Empfänger. **wahr**
- Ein Aufbewahrer im Entwurfsmuster Memento kann beliebig viele Mementos verwalten. Für die Restauration im Falle eines Reset ist er allerdings nicht verantwortlich. **wahr**
- Die Fabrikmethode sorgt dafür, dass nur eine einzige Instanz einer Klasse fabriziert wird. **falsch**
- Eine Schablonenmethode ist immer auch eine Fabrikmethode.

Wahr oder falsch?

- Bei dem Entwurfsmuster Befehl kennt der Empfänger den Befehl nicht, jedoch der Befehl den Empfänger. **wahr**
- Ein Aufbewahrer im Entwurfsmuster Memento kann beliebig viele Mementos verwalten. Für die Restauration im Falle eines Reset ist er allerdings nicht verantwortlich. **wahr**
- Die Fabrikmethode sorgt dafür, dass nur eine einzige Instanz einer Klasse fabriziert wird. **falsch**
- Eine Schablonenmethode ist immer auch eine Fabrikmethode. **falsch**

Wahr oder falsch?

- Bei dem Entwurfsmuster Befehl kennt der Empfänger den Befehl nicht, jedoch der Befehl den Empfänger. **wahr**
- Ein Aufbewahrer im Entwurfsmuster Memento kann beliebig viele Mementos verwalten. Für die Restauration im Falle eines Reset ist er allerdings nicht verantwortlich. **wahr**
- Die Fabrikmethode sorgt dafür, dass nur eine einzige Instanz einer Klasse fabriziert wird. **falsch**
- Eine Schablonenmethode ist immer auch eine Fabrikmethode. **falsch**
- Eine Komponente kann immer nur mit einem einzigen Dekorierer versehen werden.

Wahr oder falsch?

- Bei dem Entwurfsmuster Befehl kennt der Empfänger den Befehl nicht, jedoch der Befehl den Empfänger. **wahr**
- Ein Aufbewahrer im Entwurfsmuster Memento kann beliebig viele Mementos verwalten. Für die Restauration im Falle eines Reset ist er allerdings nicht verantwortlich. **wahr**
- Die Fabrikmethode sorgt dafür, dass nur eine einzige Instanz einer Klasse fabriziert wird. **falsch**
- Eine Schablonenmethode ist immer auch eine Fabrikmethode. **falsch**
- Eine Komponente kann immer nur mit einem einzigen Dekorierer versehen werden. **falsch**

- Entwurfsmuster kommen sehr sehr sehr wahrscheinlich dran!

- Entwurfsmuster kommen sehr sehr sehr wahrscheinlich dran!
- Kategorien helfen beim Lernen

- Entwurfsmuster kommen sehr sehr sehr wahrscheinlich dran!
- Kategorien helfen beim Lernen
- jedes Entwurfsmuster erfüllt einen bestimmten Zweck
⇒ nicht nur die Klassen und Methoden auswendig lernen, sondern das Prinzip verstehen

- Entwurfsmuster kommen sehr sehr sehr wahrscheinlich dran!
- Kategorien helfen beim Lernen
- jedes Entwurfsmuster erfüllt einen bestimmten Zweck
⇒ nicht nur die Klassen und Methoden auswendig lernen, sondern das Prinzip verstehen
- bei Unklarheiten in Head First Design Patterns nachlesen ;)

Aufgabe 1: Manager-Deutsch und Architekturstile

- Architekturstile nochmal anschauen

Aufgabe 1: Manager-Deutsch und Architekturstile

- Architekturstile nochmal anschauen

Aufgabe 2: Iterator für Plug-Ins

- Iterator-Muster selbst benutzen

Aufgabe 3: Geometrifly mit Entwurfsmustern

- überlegen, welches Entwurfsmuster **warum** Sinn macht

Aufgabe 3: Geomtrify mit Entwurfsmustern

- überlegen, welches Entwurfsmuster **warum** Sinn macht

Aufgabe 4: Geomtrify umstrukturieren

- Überlegungen aus Aufgabe 3 umsetzen

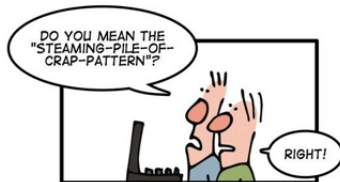
Aufgabe 5: GUI erweitern

- nochmal ServiceLoader \implies diesmal mit Primitiven

Abgabe

- Deadline am 5.7. um 12:00
- Aufgabe 1, 3 handschriftlich (wirklich handschriftlich!)

Bis dann! (dann := 10.07.17)



THE HYPE IS LONG GONE BUT
DESIGN PATTERNS ARE STILL USEFUL