Entwurfsmuster

-

http://www.philipphauer.de/study/se/design-pattern.php

Irene Rothe

irene.rothe@h-brs.de

Planung

✓ Einstiegsbeispiele: Swimmingpool (Waschmaschine), Schiffe versenken
✓ Klasse: Datei mit Eigenschaften (Attributen) und Fähigkeiten (Methoden) möglicher Objekte
✓ **OO-Basics**: Abstraktion, Kapselung, Vererbung, Polymorphie
✓ IDEs: Eclipse, javac-Editor, IntellJ
✓ Abstrakte Klassen und Interfaces
✓ **OO-Prinzipien**:

- Kapseln, was sich ändert
- Programmieren auf Schnittstelle
- Schwache Koppelung und starke Kohäsion
- Subklassen sollten ihre Superklasse vertreten können

✓ OOA: Analyse, Design, Programmierung
→Entwurfsmuster
→UML
→Heuristiken
→Parallelprogrammierung mit Java

Design der Folien

- Grün hinterlegte Informationen sind sehr wichtig und klausurrelevant.
- Alles hinter „**Achtung**“ unbedingt beachten!
-  verwende ich, wenn überraschende Probleme auftreten können.
- „Tipp“ benutze ich, um Ihnen einen Weg zu zeigen, wie ich damit umgehen würde.
- „Bemerkung“ in Folien beziehen sich meist auf Sonderfälle, die nicht unbedingt klausurrelevant sind, aber für Sie beim Programmieren eine Bedeutung haben könnten.
-  hinter diesem Symbol ist ein Link fürs Anhören bzw. Gucken weiterer Infos

Entwurfsmuster…

…sind für das Programmieren im Großen was Algorithmen für das Programmieren im Kleinen sind.

Entwurfsmuster

Zeit zum Suchen eines passenden Entwurfsmuster

<

Aufwand für Refactoring-Zyklen

Entwurfsmuster

Idee:

- Strukturierung von Klassen, die Probleme lösen sollen,
- Sammlung von bewährten Lösungen für immer wiederkehrende Problemstellungen
- Anwendung der OO-Prinzipien

Ziel:

- länger im Entwurf bleiben
- Mustervokabular (mit weniger mehr sagen)
- sind auf höherer Ebene als Bibliotheken angesiedelt

Beispiele:

- Schiffeversenken: Umsetzung des *Strategy*-Musters, *Template*-Musters
- Witzeerzähler: Umsetzung des *Beobachter*musters
- Türme von Hanoi: Umsetzung des *Kompositum*muster (Container)

Prägung von der **Gang of Four** (GOF): Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides.

Entwurfsmuster

Vorteile:

- Standardisierung des Entwicklungsprozesses
- Diskussion über Softwareentwickler auf abstrakter Ebene
- Idee im Muster erkennbar
- Realisierung der Vorteile von OO
- Wiederverwendung von Code
- Ideenlieferant
- Lernen von erfahrenen Programmierung mit langjähriger Praxis
- Leckerbissen für Programmierfans

Nachteile:

- aufwändige Konzeption
- hohe Einarbeitungszeit

Einteilung von Mustern:

→ Strukturmuster
→ Verhaltensmuster
→ Erzeugungsmuster

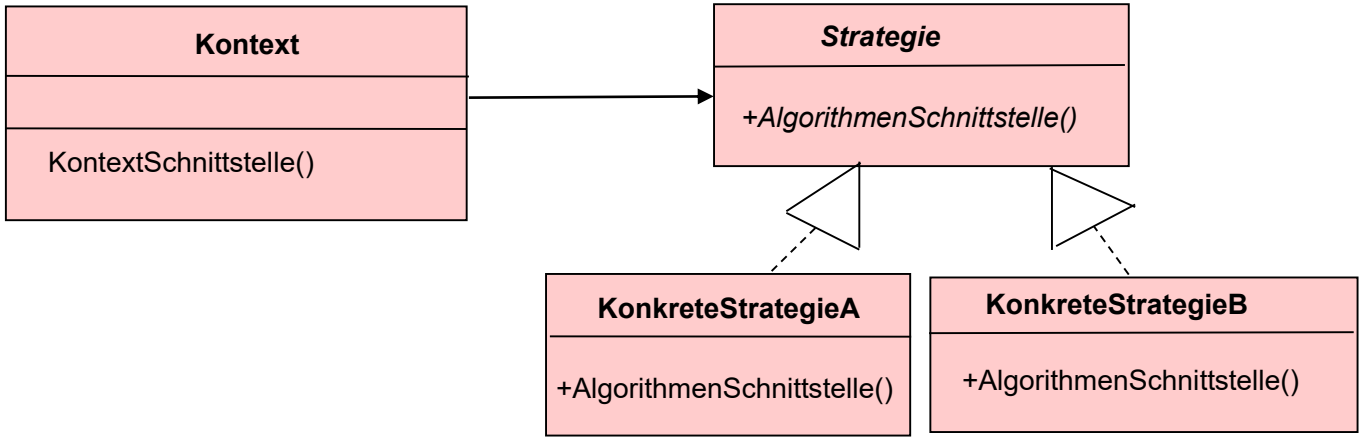
*Strategie*muster

Motivation: Umsetzung der/des Heuristik/OO-Prinzips „Komposition ist besser als Vererbung“

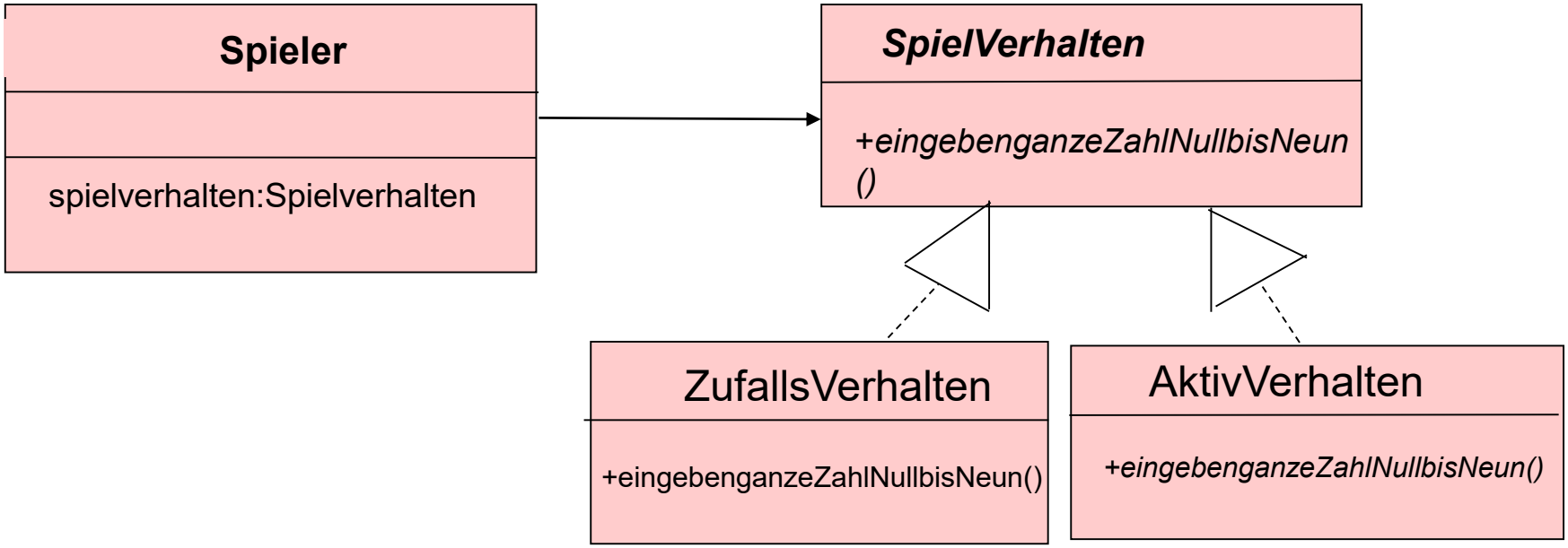
Idee: Algorithmenkapselung, damit sie austauschbar sind (eventuell zur Laufzeit)

Vorteile:

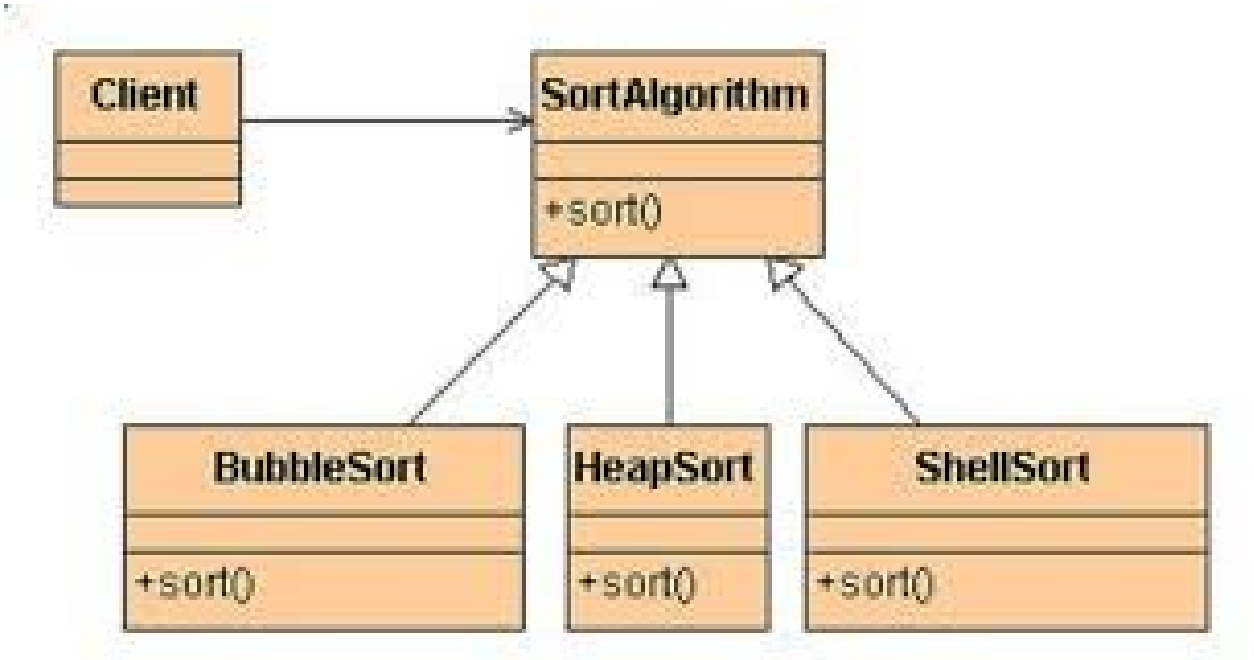
- Definition eines Pools von Algorithmen möglich
- einheitliche Schnittstelle
- Aufruf in immer gleicher Art



Strategiemuster - Beispiel Schiffe versenken



Strategiemuster – Beispiel Sortieren

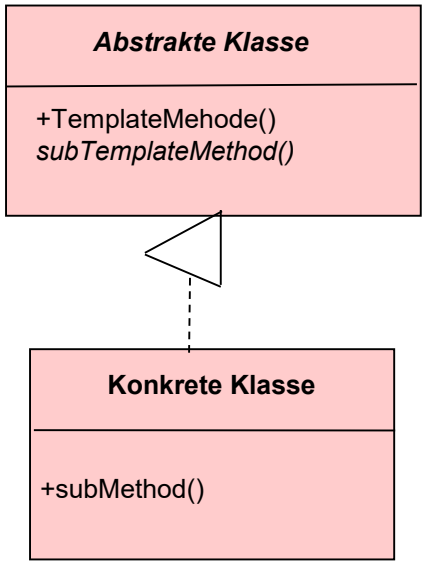


Template Method Muster

Motivation: Skelett eines Algorithmus oder Objektes definieren, aber die konkrete Ausformung an Unterklassen delegieren.

Vorteile:

- Definition eines Pools von Algorithmen möglich
- einheitliche Schnittstelle
- Aufruf in immer gleicher Art
- Teilcodierung vorhanden



Beispiele:

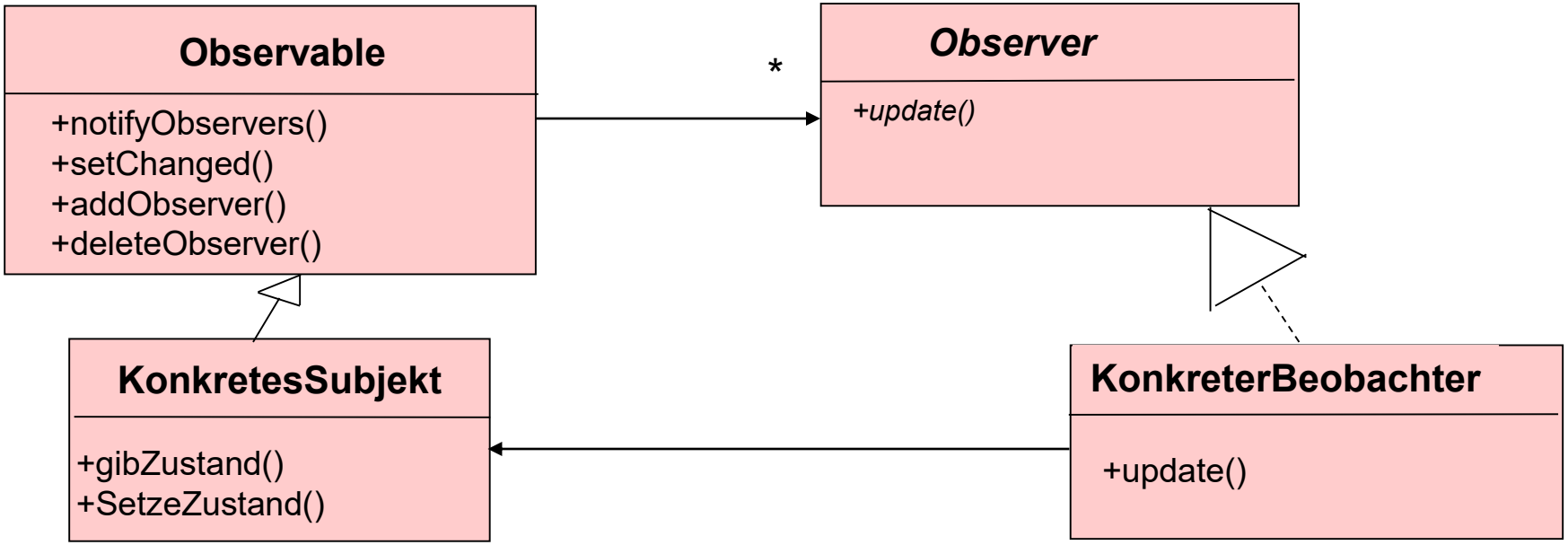
- Spieler-Klasse
- Spiel-Klasse: Initialisierung, Start, Spielen, Ende

*Beobachter*muster

Idee: ein Objekt wird beobachtet + eine Liste von Objekten, die dieses beobachten (sehr nützlich bei GUIs, ist auch bekannt unter Listener)

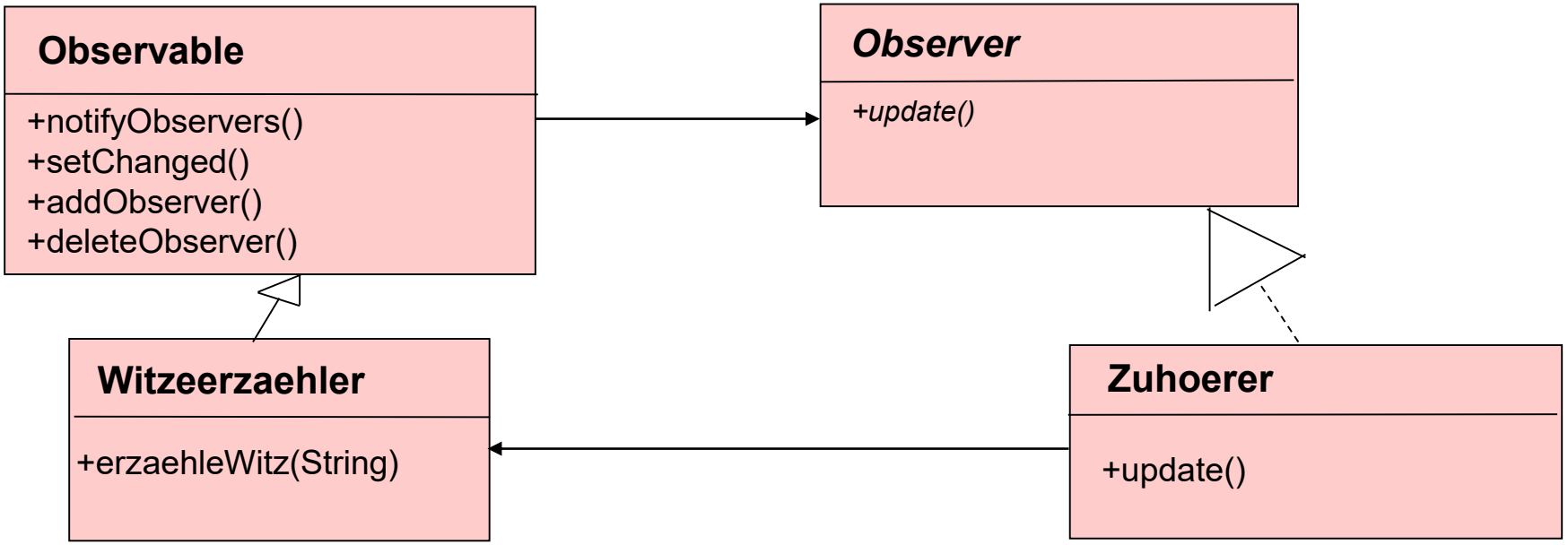
Vorgehen:

1. Interessierte Beobachter melden sich beim **Observable-Objekt** an
2. Ändert sich das **Observable-Objekt**, benachrichtigt es seine „Fans“
3. Jeder Beobachter hat die Methode **update** implementiert, was durch das Interface **Observer** verlangt wird



*Beobachter*muster

Beispiel: Witzeerzähler-Beispiel umgesetzt in Java als **`Observable`** (import java.util.Observable; import java.util.Observer;)

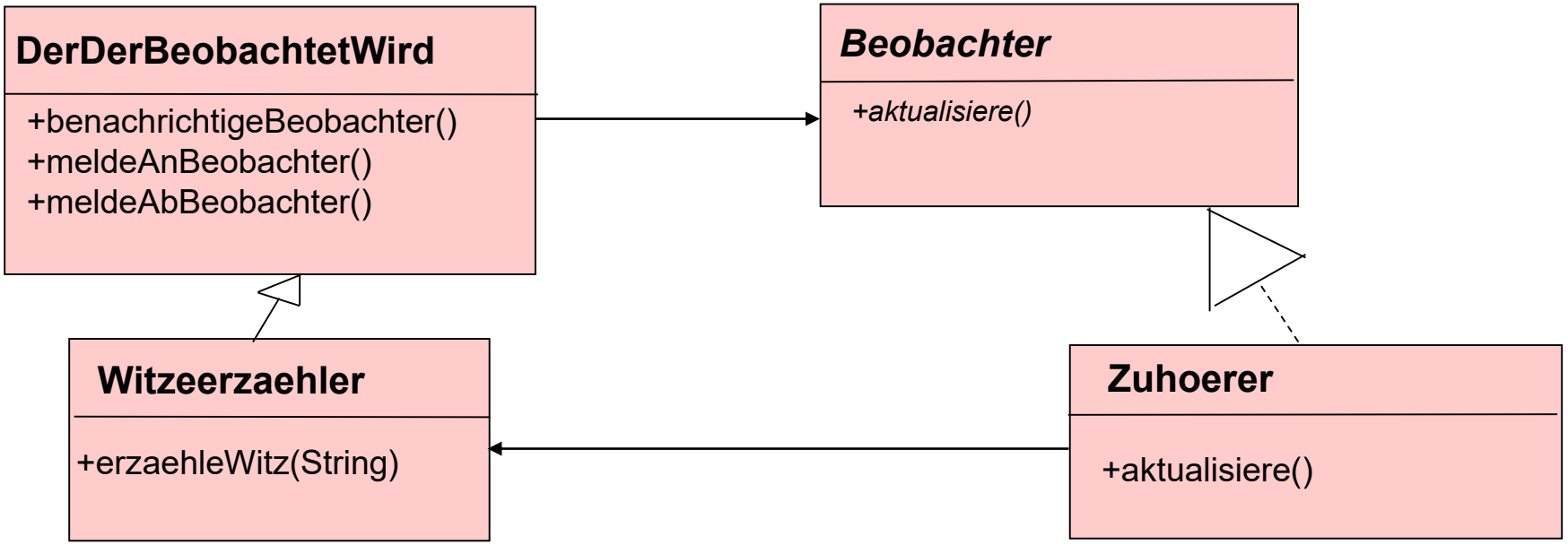


Besondere Idee: der **`Witzeerzaehler`** ruft in **`erzaehleWitz()`** **`notifyObserver`**() auf (deklariert in Superklasse), in **`notifyObserver()`** wird **`update()`** der **`Zuhoerer`** aufgerufen, was nur geht, weil in **`Observable`** eine Liste aller **`Zuhoerer`**-Instanzen gespeichert ist.

Beobachtermuster - Witzeerzähler

```
class Witzeerzaehler extends Observable{
        public void erzaehleWitz(String witz){
                setChanged();
                notifyObservers(witz);//Aufruf von update
}}
class Zuhoerer implements Observer{
        private String name;
        Zuhoerer(String name){this.name=name;}
        public update(Observable o, Objekt obj){
          System.out.println(name+" lacht ueber den Witz '"+obj+"'.");
}}
class Party{
        public static void main(){
                Zuhoerer max = new Zuhoerer("Max");
                Zuhoerer susi = new Zuhoerer("Susi");
                Witzeerzaehler irene = new Witzeerzaehler();
                irene.addObserver(max);
                irene.addObserver(susi);
                irene.erzaehleWitz("Was ist gelb und kann schiessen?
                                    Eine Banone.");
                irene.deleteObserver(max);
                irene.erzaehleWitz("Treffen sich zwei Jaeger");
                irene.deleteObserver(susi);
}}
```

*Beobachter*muster - selbst gemacht



Bemerkung: Beispiel in Java zeigen: Ordner BeobacherMuster, Thomas Breuer: Skript S.32 (Timerinterrupt)

*Kompositum*muster

Idee:

- Fügt mehrere Objekte zu einer Baumstruktur zusammen und ermöglicht es, diese wie ein einzelnes Objekt zu verwenden.
- Repräsentiert Teil-Ganzes-Hierarchien

Vorgehen:

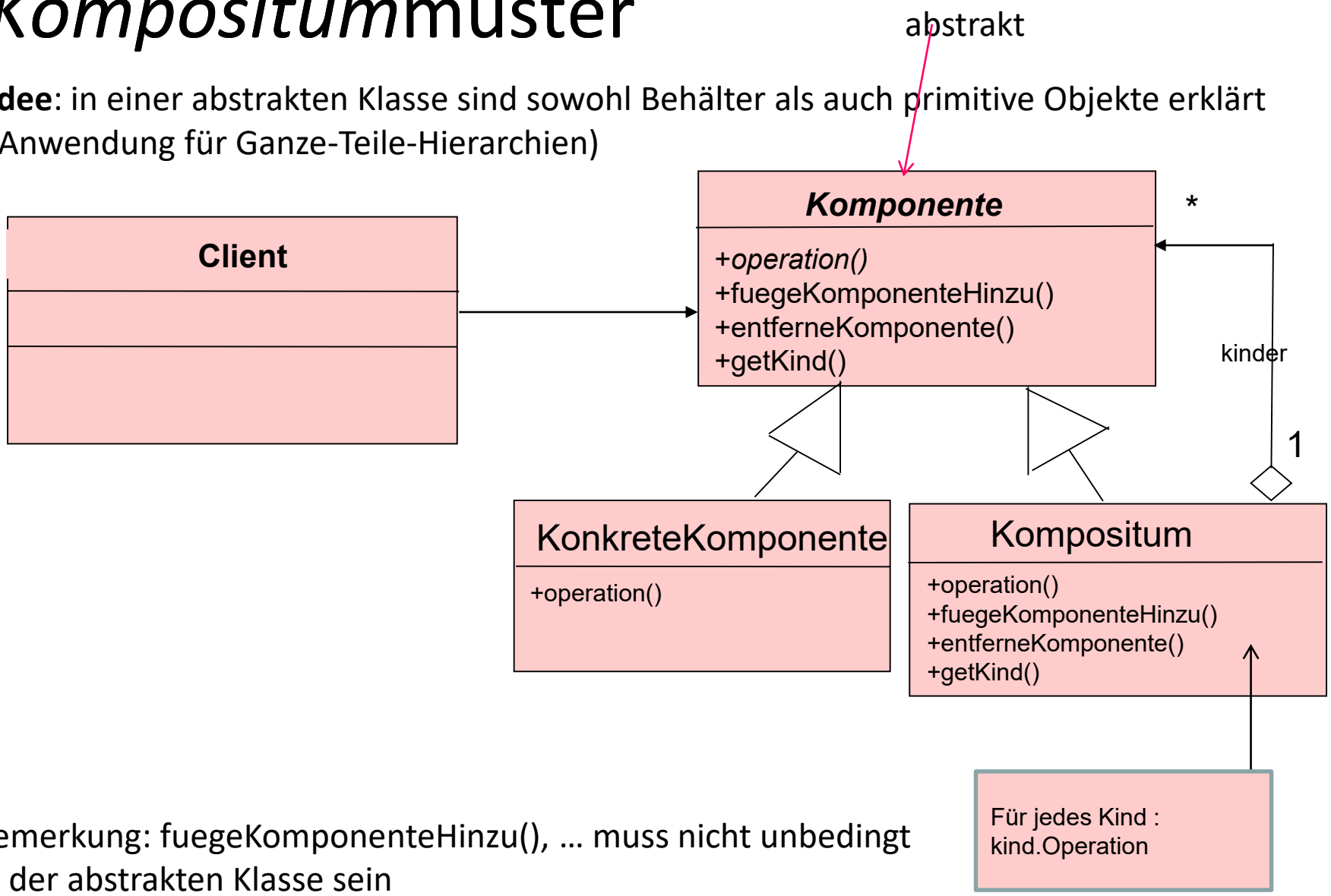
- in einer abstrakten Klasse sowohl primitive Objekte als auch ihre Behälter repräsentieren
- somit können sowohl einzelne Objekte als auch ihre Kompositionen einheitlich behandelt werden.

Beispiele:

- Javas AWT-Klassen sind nach dem Kompositum-Muster gebaut. Da alle von Container erben, können sie jeweils selbst wieder Elemente aufnehmen.
- Man kann gut rekursive Strukturen so aufbauen

*Kompositum*muster

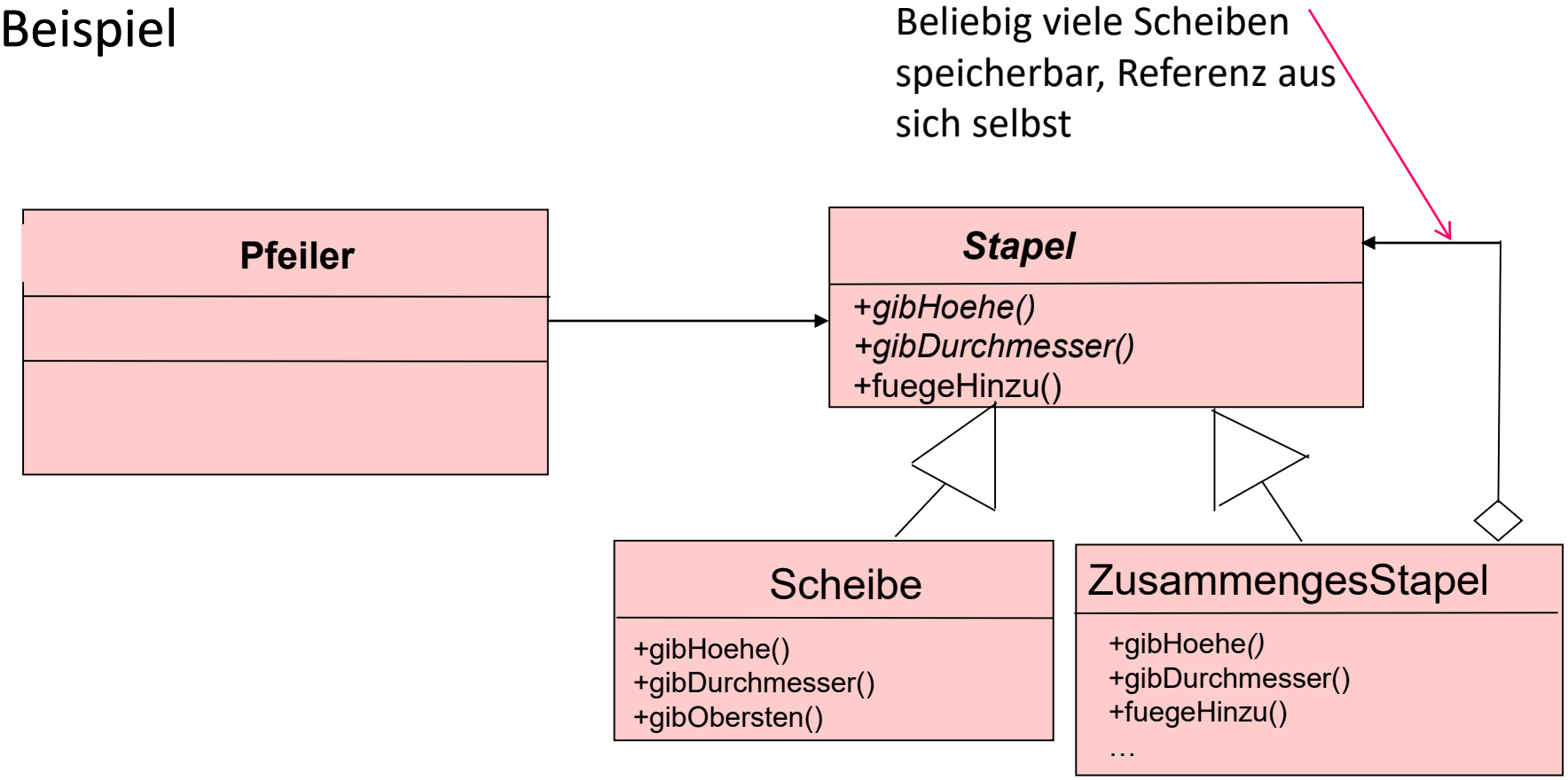
Idee: in einer abstrakten Klasse sind sowohl Behälter als auch primitive Objekte erklärt (Anwendung für Ganze-Teile-Hierarchien)



Bemerkung: fuegeKomponenteHinzu(), … muss nicht unbedingt In der abstrakten Klasse sein

Kompositum: Türme von Hanoi

Beispiel



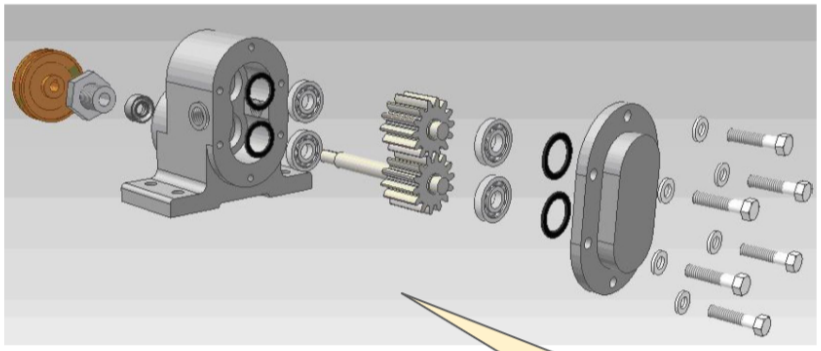
„Teil von“, lebt aber auch unabhängig (Ehe/Ehepartner)

Kompositum: weiteres Beispiel 1a

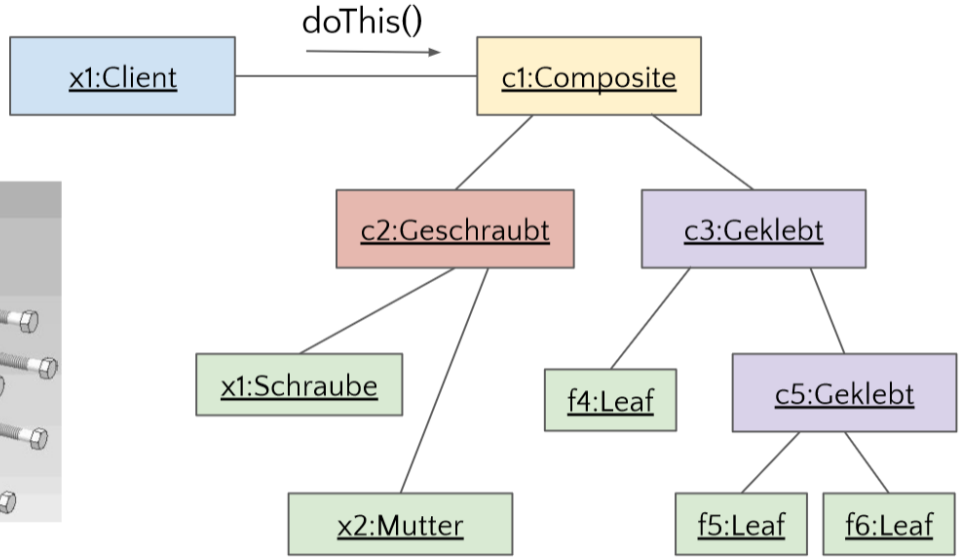
Kompositum (*Composite*)

EN: ˈkɒmpəzɪt;

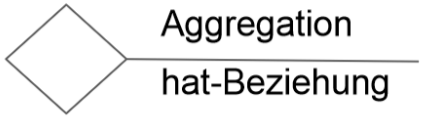
US: kɒmˈpɒzɪt;



Quelle: Wikipedia
https://commons.wikimedia.org/wiki/File:Bomba_de_engranajes3.JPG#

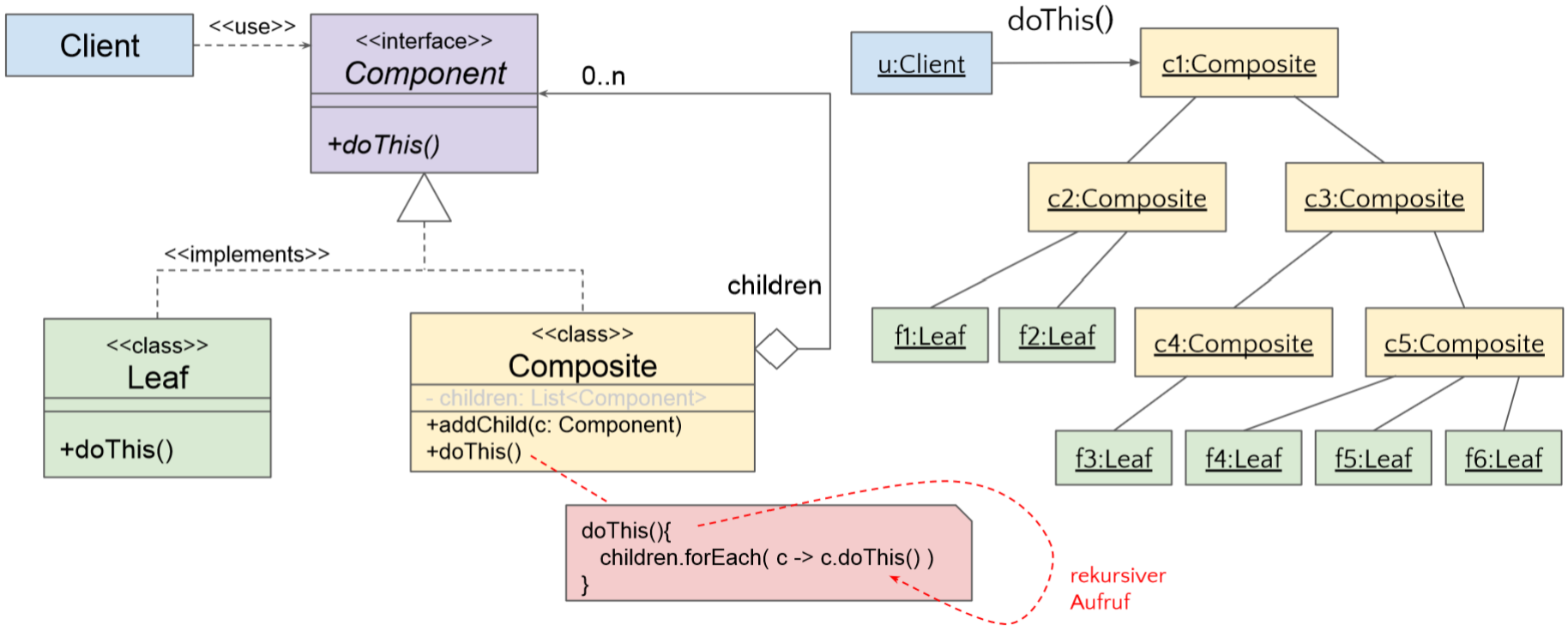


Ein Objekt **hat** viele Einzelteile.



Kompositum: weiteres Beispiel 1b

Kompositum (*Composite*)

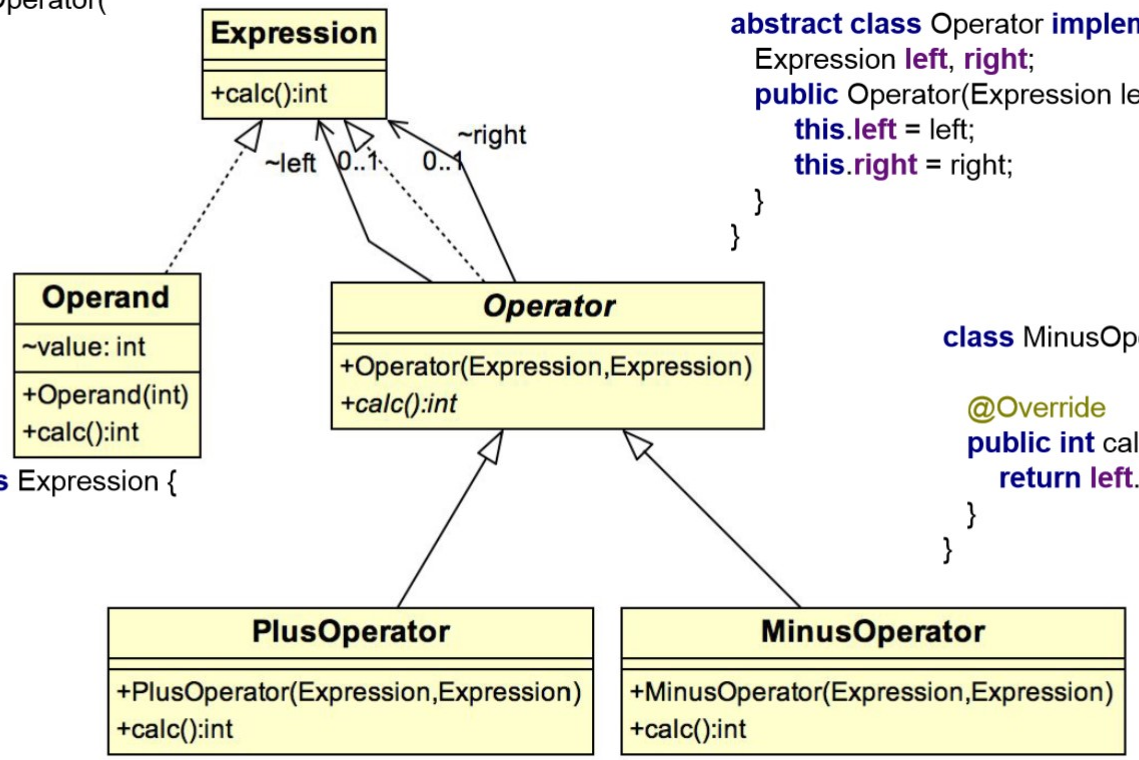


Kompositum: weiteres Beispiel 2

Muster
Composite

Arithmetische Ausdrücke als Kompositum (*Composite*)

```
Expression ex = new PlusOperator(
      new MinusOperator(
            new Operand(3),
            new Operand(2)
      ),
      new Operand(1)
);
assertEquals(2, ex.calc());
```




```
abstract class Operator implements Expression {
  Expression left, right;
  public Operator(Expression left, Expression right){
     this.left = left;
     this.right = right;
  }
}
```

```
class MinusOperator extends Operator {

  @Override
  public int calc(){
     return left.calc() - right.calc();
  }
}
```

```
class Operand implements Expression {
  int value;

  @Override
  public int calc() {
     return value;
  }
}
```

Kompositum - Test

Composite

… mit JUnit getestet

```
public interface Expression {
  int calc();
  void addChild(Expression child);
}
```

```
public class Operand implements Expression {
  int value;
  public Operand(int value){
      this.value = value;
  }

  @Override
  public int calc() {
     return value;
  }

  @Override
  public void addChild(Expression child) {}
}
```

JUnit

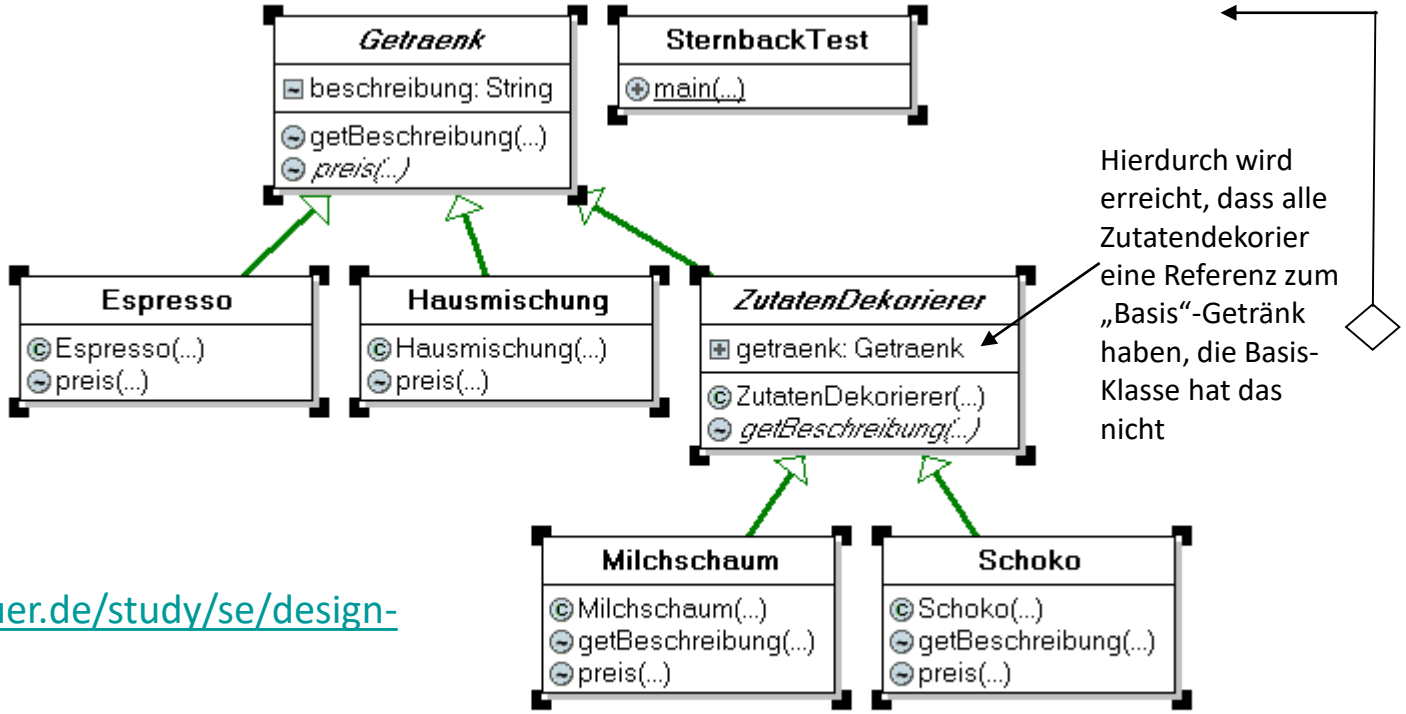
```
public class MinusOperatorTest {
  Expression ex;

  @Before
  public void setUp() throws Exception {
     ex = new PlusOperator(
           new MinusOperator(
                 new Operand(333),
                 new Operand(222)
           ),
           new Operand(11)
     );
  }

  @Test
  public void testCalc() throws Exception {
      assertEquals(122, ex.calc());
  }
}
```

Decoratormuster

Motivation: Umsetzung des OO-Prinzips „Klassen offen für Erweiterung, geschlossen für Veränderung“

Idee: Dekorierer haben gleichen Supertyp wie die Objekte, die sie dekorieren

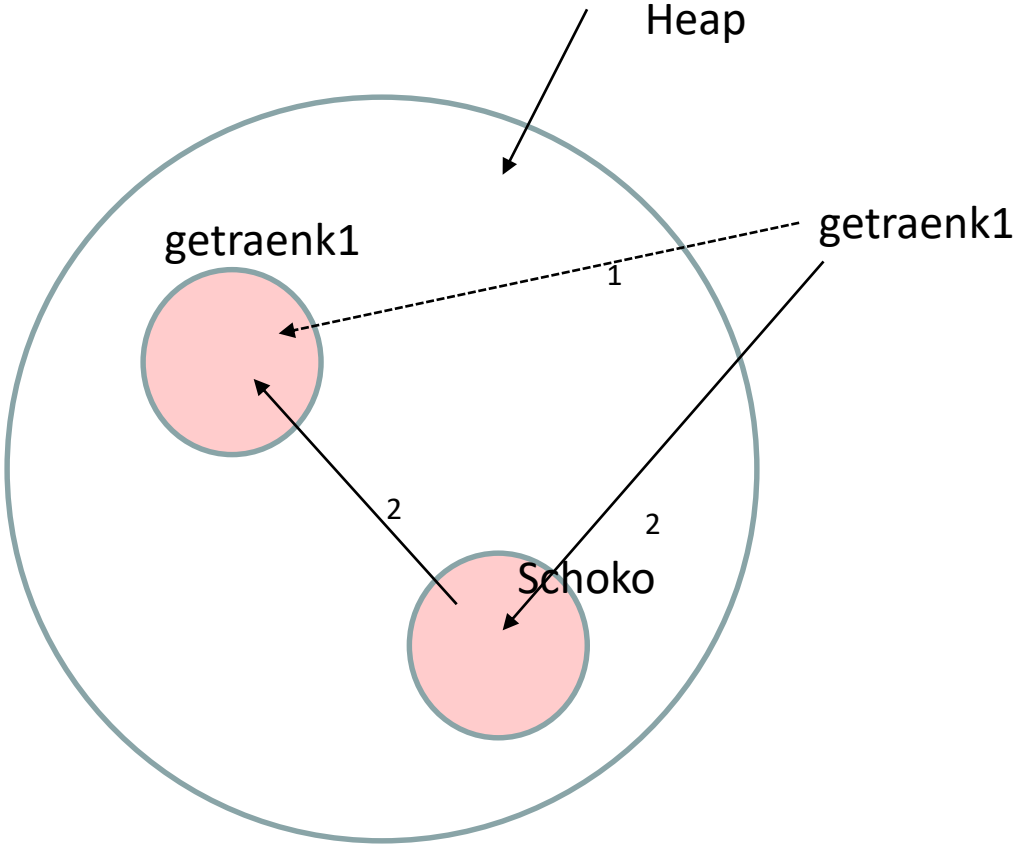
Vorgehen: Vererbung nutzen, um Übereinstimmung zu erhalten.



http://www.philipphauer.de/study/se/design-pattern/decorator.php

Bemerkung: Unterschied zum Kompositummuster: dort gibt es keine Zwischenklasse, wie *Zutatendekorierer*

*Decoratorm*uster (Objektdiagramm)



*Decoratorm*uster – Beispiel Stoppuhr (Breuer)

Wo könnte man hier das Decoratormuster nutzen?

*Decoratorm*uster – Beispiel Stoppuhr (Breuer)

Wo könnte man hier das Decoratormuster nutzen?

→ ButtonKombi?
→ Untere Schaltfläche besteht aus so einer Kombi, 3 Felder, wo man touchen kann

WiSe22

Delegation - Beispiel aus der realen Welt

→ wurde genutzt im Decorator Muster (Boss~Schoko.java)

```
interface Worker{
  Result work();
}
class Secretary implements Worker{
  Secretary(){}
  Result work(){
    Result myResult = new Result();
    return myResult;
  }
}
class Boss implements Worker{
   private Secretary secretary;
   Boss(){}
   Result work(){
     return secretary.work();
   }
}
```

Boss dekoriert
sich mit Sekretärin

*Singleton*muster

Idee: sichert, dass es nur eine Instanz einer Klasse gibt, bietet globalen Zugriffspunkt für diese eine Instanz

```
public class SingletonBeispiel {
    private static SingletonBeispiel instanz = null;
    private SingletonBeispiel() {}
    public static SingletonBeispiel getInstanz() {
      if (instanz == null) {
         instanz = new SingletonBeispiel();
      }
      return instanz;
    } }
```

Hier wird Klasse benutzt und nicht Variable logfile.

Fakten:

- Klasse erzeugt selbst Instanz, verwaltet diese und ermöglicht Zugriff
- **`private`** Konstruktor verhindert, dass eine Instanz der Singleton-Klasse aus einer anderen Klasse heraus erstellt werden kann
- Beispiel: Timer, Logfile (es soll nur eins davon geben, alle schreiben in diese Datei rein), erstmaliges Nutzen so einer Instanz z.B.:
  ```
  SingletonBeispiel logfile = SingletonBeispiel.getInstanz();
  ```
- **Achtung** bei Multithreading! → https://www.theserverside.de/singleton-pattern-in-java/

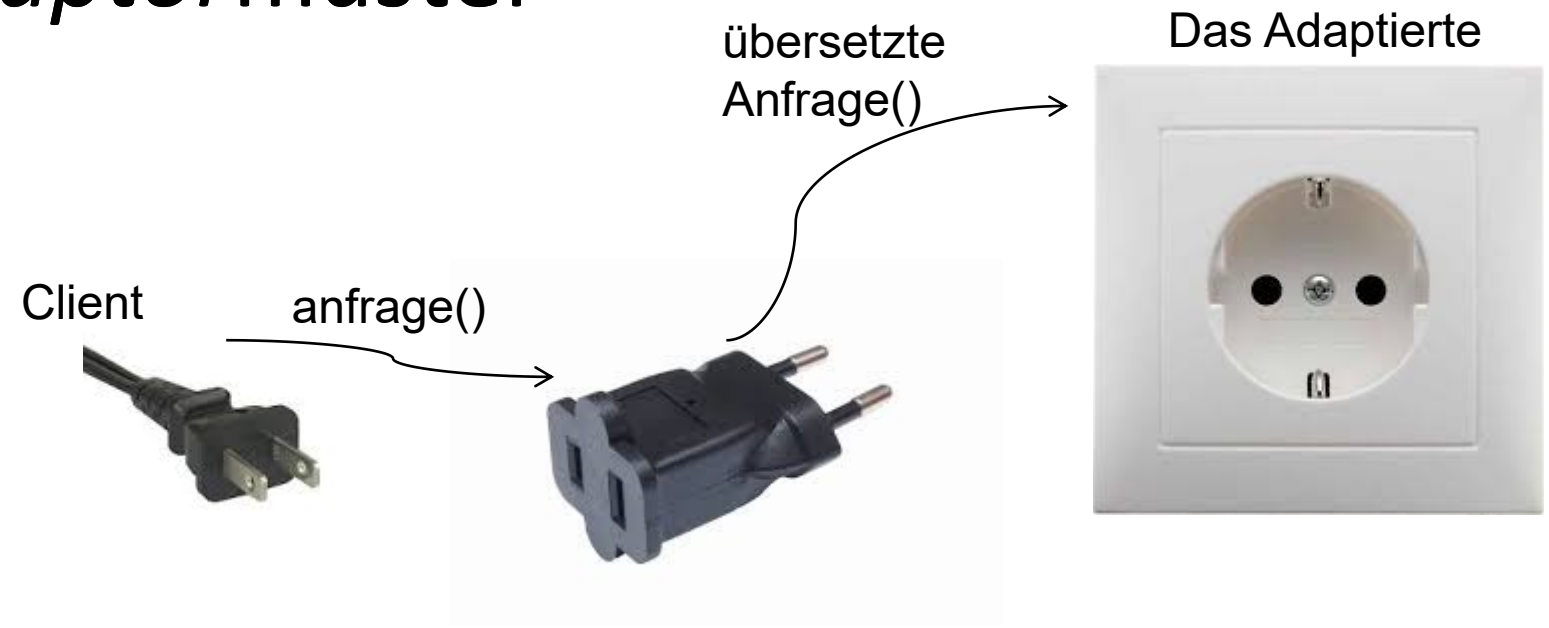
*Adaptor*muster

Idee: Schnittstelle in eine andere umwandeln

Keine Änderung!

Keine Änderung!

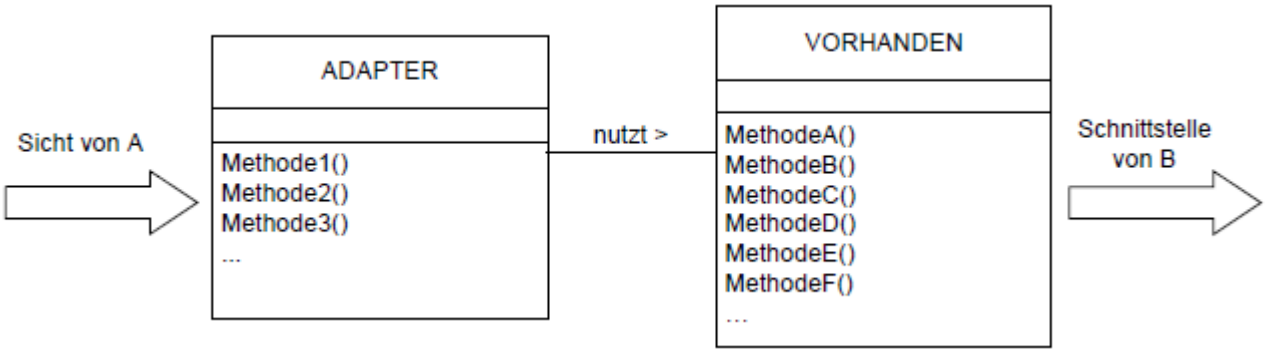
*Adaptor*muster



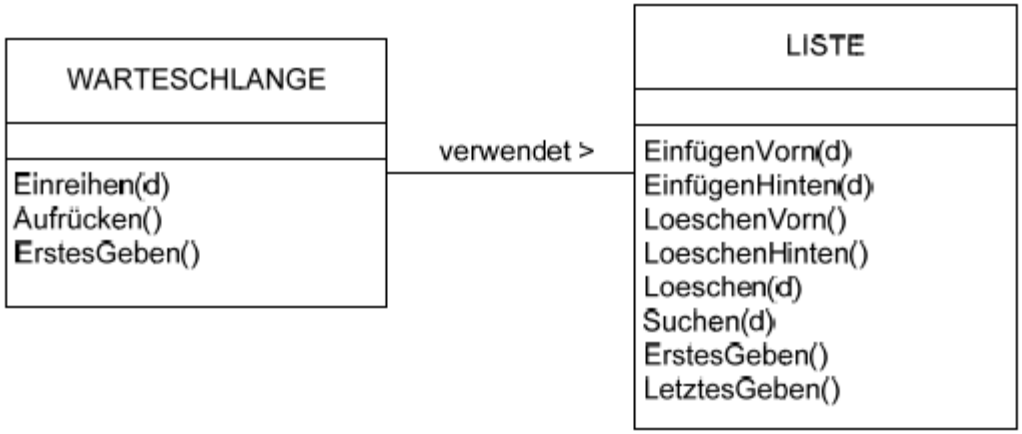
Implementiert die Ziel-Schnittstelle und hält eine Instanz des adaptierten Objekts

*Adaptor*muster

Idee: Bindeglied zwischen inkompatiblen Schnittstellen
Vorgehen: eine vorhandene Klasse wird auf eine neue Situation angepasst



*Adaptor*muster - Beispiel: Warteschlange



Die Methoden der Warteschlange verwenden einfach die korrespondierenden Methoden der Klasse LISTE: Einreihen nutzt EinfuegenHinten, Aufruecken nutzt LoeschenVorn und ErstesGeben nutzt ErstesGeben. Alle für eine Warteschlange nicht anwendbaren Methoden sind nun nicht mehr sichtbar.

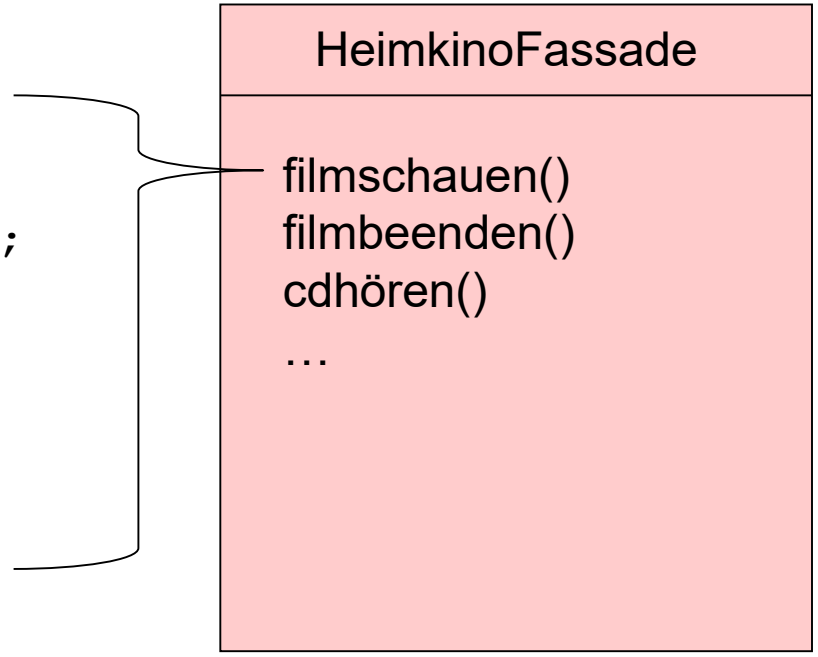
```
class WARTESCHLANGE {
  private LISTE liste;
  WARTESCHLANGE () {
     liste = new LISTE();
  }
  void Einreihen(DATENELEMENT d) {
     liste.EinfuegenHinten(d); }
  ...
}
Geklaut bei:
http://wvsg.schulen2.regensburg.de/joomla/images/Faecher/Informatik/Informatik_11/informatik_11_4_3_Entwurfsmuster.html
```

*Facade*muster

Idee: eine Schnittstelle, die mehrere komplexe Schnittstellen zusammenfasst und dabei eine Vereinfachung erzeugt; wird genutzt bei steigender Anzahl der Features

Film im Heimkino gucken:

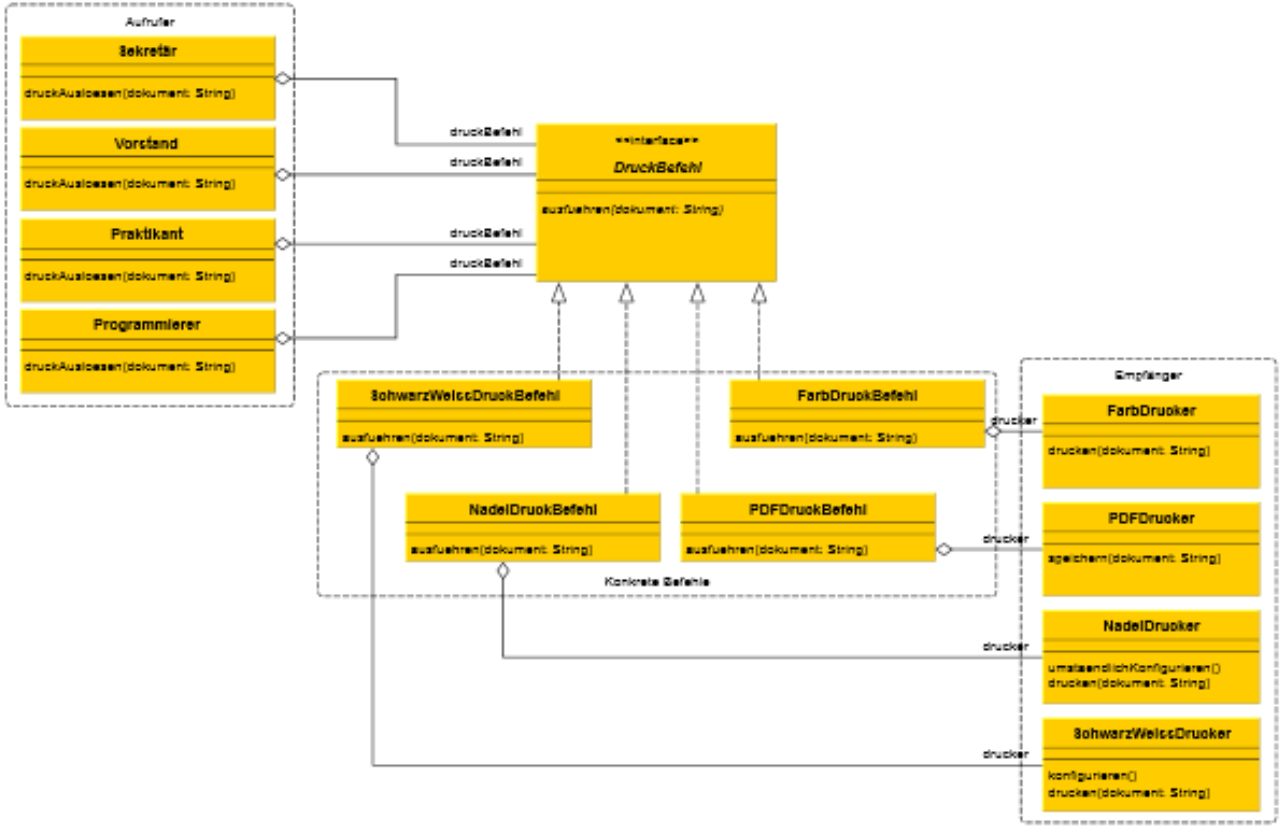
```
popcorn.rein();
popcorn.starten();
beleuchtung.dimmen();
leinwand.runter();
beamer.an();
verstaerker.an();
dvd.rein();
dvd.spielen(film);
```



Methoden kommen aus 6 verschiedenen Klassen (Popcornmaschine, Beleuchtungstechnik, Leinwandtechnik, Beamer, DVDLaufwerk, Tontechnik)

*Command*muster

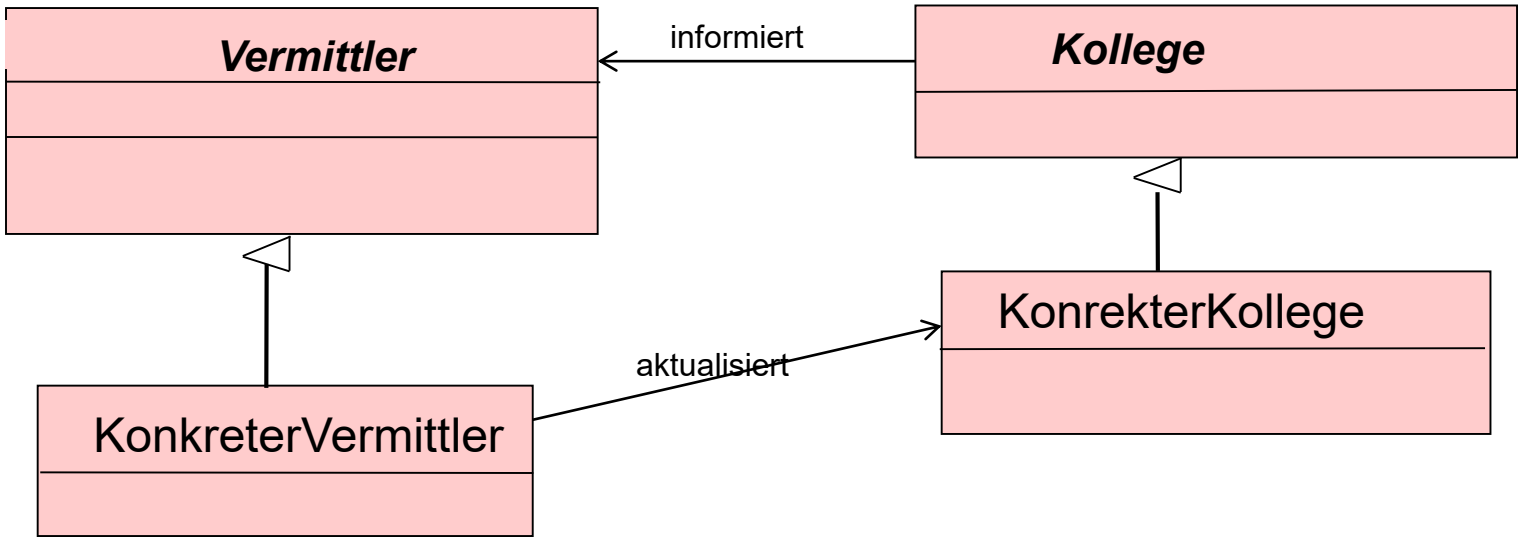
Idee: man spricht nur mit einer Schnittstelle, den konkreten Befehl muss niemand wissen, nur der Befehl selbst muss wissen, was zu tun ist.



https://www.philipphauer.de/study/se/design-pattern/command.php

*Mediator*muster

Idee: Vermittler, der unüberschaubere Beziehungen bündelt
Fakten: alle Kommunikation läuft über den Vermittler, er kontrolliert und koordiniert die Interaktion von Objekten



Abstrakte Vermittlerklasse kann weggelassen werden, wenn *ein* Vermittler reicht.
Beispiel: Verwaltung bei unseren Spielen

Entwurfsmuster…

- sind gut, wenn man etwas programmieren will und keine so richtige Ahnung hat wie. Dann sucht man nach einem ähnlichen Beispiel. Ist dabei ein Entwurfsmuster umgesetzt worden, kann man ziemlich sicher sein, dass die Idee sich bewährt hat und man den Code sehr gut als Grundlage nutzten kann.
- Das ganze Muster muss dann nicht unbedingt komplett verstanden werden, solange man weiß, wie man es benutzen muss, um das eigene Problem zu lösen.
- Das Motto ist ein bisschen wie folgt: Hauptsache es funktioniert!

Entwurfsmuster

Zeit zum Suchen eines passenden Entwurfsmuster

<

Aufwand für Refactoring-Zyklen

Entwurfsmusterkatalogaufbau

Name des Musters

Problem: Kontext, wann das Muster anwendbar ist

Lösungsmuster einer **Lösung** mit allen Komponenten:
kein konkreter Entwurf, keine Implementierung, aber eine Schablone für diese

Konsequenzen: Ergebnisse und Trade-offs bei Anwendung

Beispiel

MVC (ModellViewController)-Konzept

- Kombination von Beobachtermuster, Strategie und Kompositummuster
- Teilweise in Bibliothek vorhanden, halb vorprogrammiert
- Beispiele: Abonnieren eines Blogs: man wird informiert, wenn es Neuigkeiten gibt, man kann Abo kündigen
- Kommunikation kann sein: ftp, email, thread …

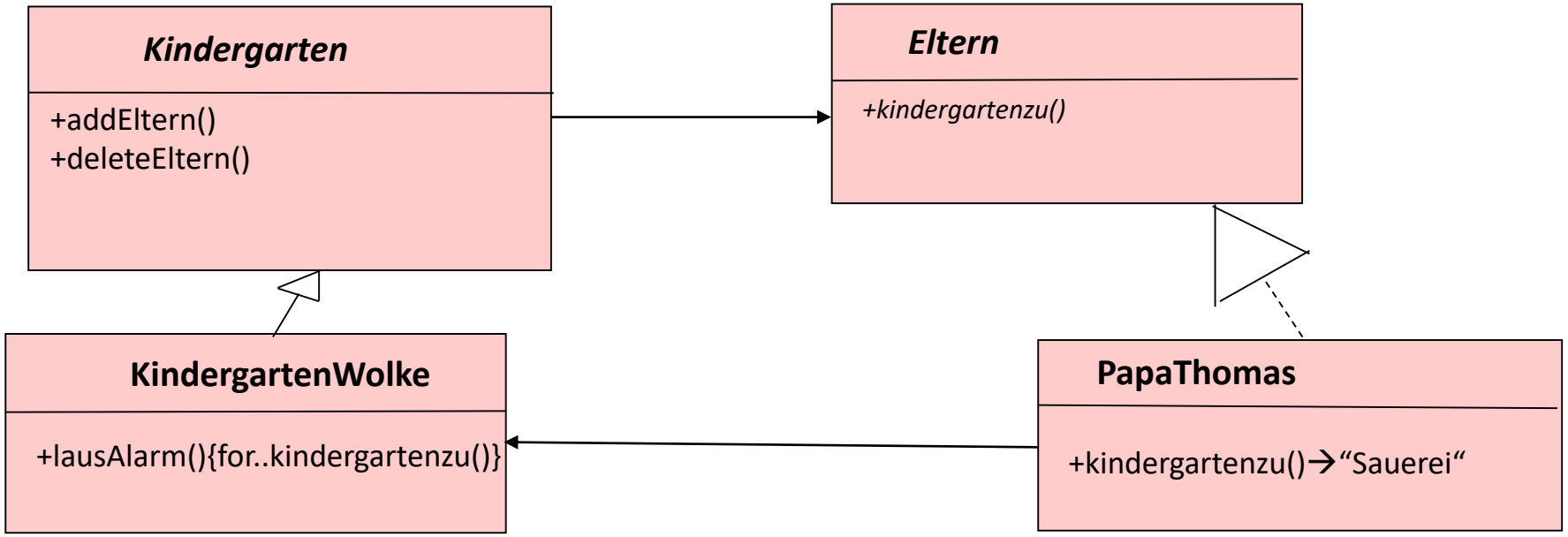
Idee: Schichtenmodell

- Datenschicht: Speicherung der Daten → **Model**
- Logikschicht: Ausführung von Prozessen, Analysieren der Daten → **Controller**
- View-Schicht: Darstellung der Daten → **View**
- Command-Control-Schicht: Interaktion mit dem Benutzer

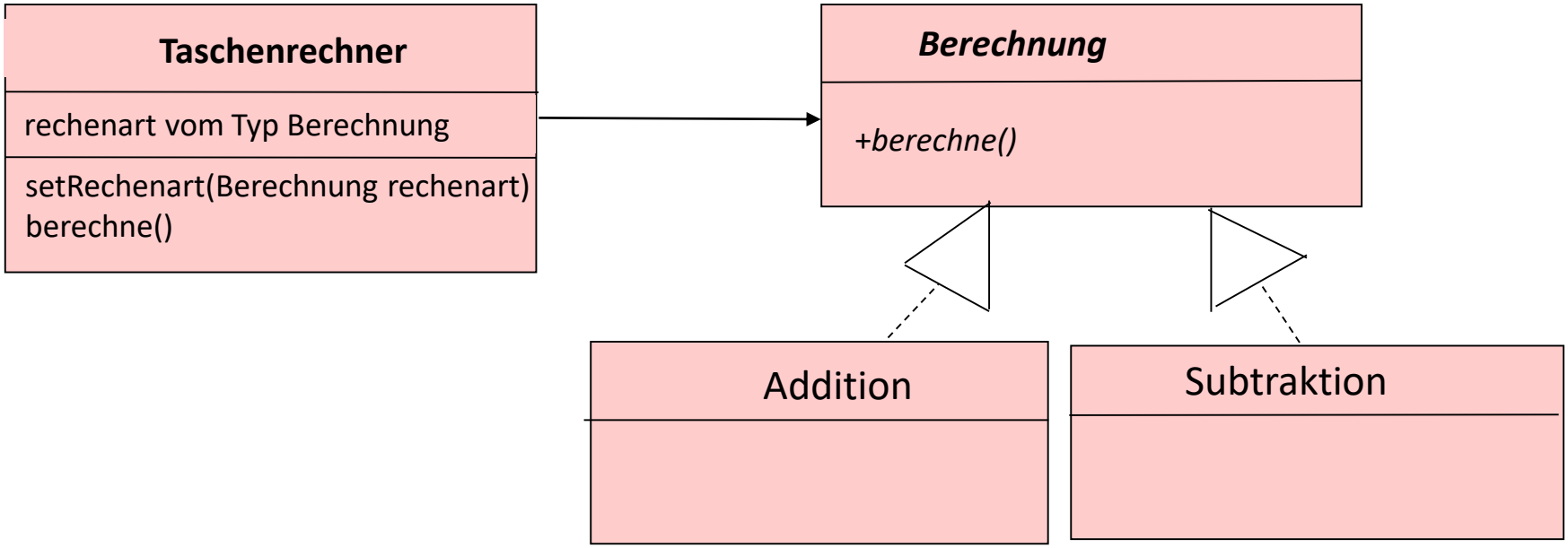
Übung - Welches Muster wäre passend?

- Steuerberechnungsprogramm für die EU mit unterschiedlichen Steuersätzen
- Handynummernvergabe
- Zeitungen reagieren, wenn bestimmte Politiker etwas sagen
- Urlaubsreise: Schiffstour, Aufschläge für Fensterkabine, Landgänge, Bad mit Wanne
- Wie könnte man folgende Klassen anordnen? Grafik, Kreis, Rechteck, Linie, Bild; Frage: Was soll damit gemacht werden? Wie sollen sie benutzt werden?

Übung - Welches Muster?



Übung - Welches Muster?



Übung - Welches Muster?

Kreis1
Mittelpunkt Radius
get/set

Kreis2
Mittelpunkt Durchmesser
get/set

das hab ich → das will ich

?

Lebensweisheit

Übung - Zeigeruhr

- Was könnten Objekte sein?
- Welches Muster könnte passen?
- Kann das chatgpt programmieren mit Zeigern aus Zahlen?

Literatur

- „Seminararbeit“ Seng und Thelen
- „Objektorientieres Analyse&Design“ McLaughlin, Pollice, West, O‘Reilly, 2007
- „Design Patterns: Elements of Reusable Object-Oriented Software“ Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides,1994
- Tolle Webseite: http://www.philipphauer.de/study/study.php