

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

## Softwaretechnik

### Kapitel 5



1. Wieso Software Engineering?
2. Vom Einzelkämpfer zum Großprojekt

#### **Systematische Softwareentwicklung**

3. Anforderungen und Test: Basis des Projekts
4. Entwurf: Strukturen und nicht-funktionale Eigenschaften
- 5. Entwürfe notieren mit UML: Modelle im SE**
6. Design Patterns: Entwurfserfahrungen nutzen
7. Management: Technik und Projektmanagement



Leibniz Universität Hannover

SWT 2015/16 · 175

## Softwaretechnik

### Inhalt von Kapitel 5



### Systematische Softwareentwicklung

#### 5. Entwürfe notieren in UML – Modelle im Software Engineering

- Historie der UML
- Objektorientierung: Konzepte und Denkweise
- Klassendiagramme
- Sequenz- und Kollaborationsdiagramme
- Information Hiding

Leibniz Universität Hannover      SWT 2015/16 · 176

Der letzte Abschnitt, Information Hiding, fällt ein wenig aus dem Rahmen. Denn hier geht es darum, was man anderen im Team mitteilen will – mit UML oder mit anderen Mitteln.

Information Hiding ist extrem wichtig, auch ohne UML.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



## Wozu grafische Notation (2D)?

- Grafische Notation := 2-dimensionale Syntax +Semantik +Pragmatik
- Modelltheorie
  - Irrelevante Details ausblenden
  - Relevante Eigenschaften hervorheben
  - Abundante Eigenschaften günstig gestalten
    - Leicht zu verstehen
    - Leicht zu ändern
    - Menschlicher Kognition entgegenkommen: Patterns, 2D >> 1D (Text)
- Spektrum von Formalitätsgraden
  - Mehr oder weniger viele relevante Eigenschaften
  - Anfangs wenige, Grobstruktur schnell erstellen
  - Schritt für Schritt mehr, Detail ausfeilen, näher zum Code

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 · 177

UML ist im wesentlichen grafisch, also 2D.

Der Blick auf die Modelltheorie klärt wieder vieles: Was zeigt ein solches Modell eigentlich? Wer will was damit anfangen?

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



## Was ist UML?

Unified Modeling Language

**Sammlung vorwiegend grafischer Sprachen [mehrere!]**

- zur Erstellung von Anforderungs- und Entwurfsmodellen
- aus verschiedenen Perspektiven

**UML-Spezifikation oder UML-Modell =<sub>Def</sub>**  
Sammlung von UML-Diagrammen,  
die sich inhaltlich ergänzen und überlappen

- Klassenmodell spezifiziert strukturellen Aufbau des Systems
  - Bei Anforderungen zeigt es folgende relevante Originalaspekte:
    - Gegenstände der Realität, mit denen das System umgehen muss
  - Bei Entwurf und Implementierung zeigt es:
    - Klassen der Lösung

Kurt Schneider                      Leibniz Universität Hannover                      SWT 2015/16 · 178

Zu UML gehören also verwirrend viele Diagrammarten.

Ein UML-Modell bezeichnet eine Konfiguration zusammengehöriger UML-Diagramme.

Oft werden auch die Diagramme schon „UML-Modell“ genannt.

Daher ist es klarer, von der gesamten Konfiguration als „UML-Spezifikation“ zu sprechen.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

**Klassen und Objekte: Eine Denkweise**  
Genauer hingesehen

Eine **Klasse** ist die gemeinsame Beschreibung

- des Verhaltens und
- der Attribute

aller Objekte dieser Klasse

**Klasse beschreibt:**

- Verhaltensmuster
- Attribute pro Objekt
- Bezeichner

**Objekte**

- sind verschiedene Individuen
- haben je eigenen Zustand
- zeigen Verhalten:  
sie versenden Nachrichten  
und reagieren auf Nachrichten  
anderer Objekte

Jedes Objekt hat

- Dieselben Bezeichner
- Evtl. verschiedene Zustände
- Evtl. verschiedene Attributwerte

Folglich evtl. verschiedene Situationen  
im selben Verhaltensmuster

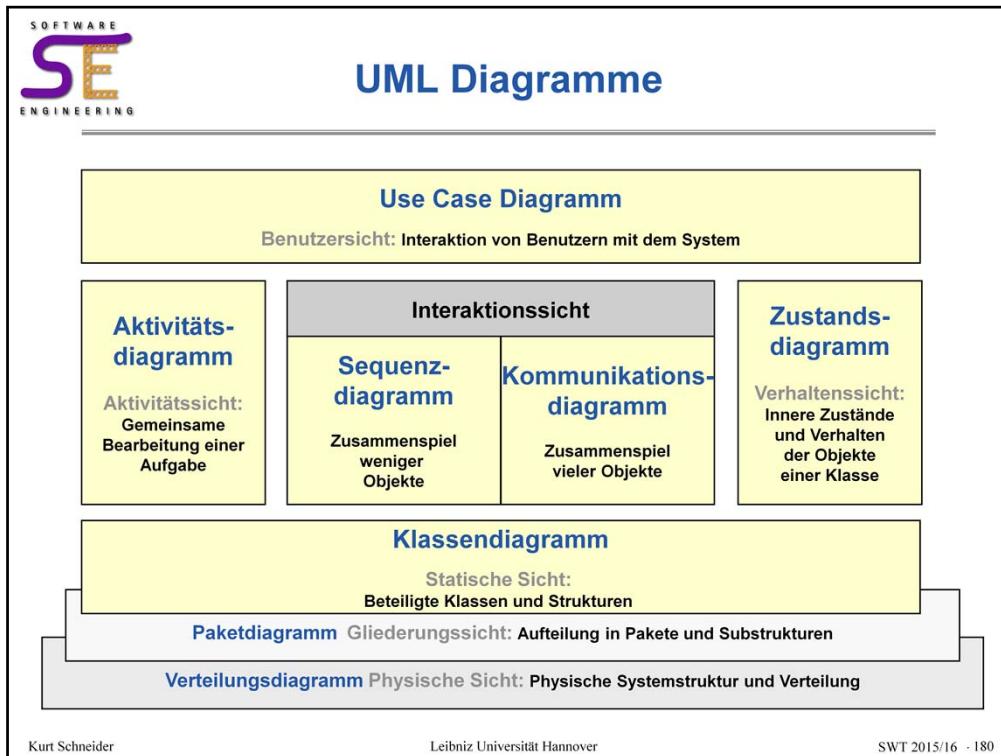
Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 179

UML ist nicht die grafische Darstellung einer normalen Programmiersprache, sondern von Konzepten, die sich durch verschiedene Programmiersprachen umsetzen lassen.

Allerdings ist UML auf Objektorientierung ausgelegt. Sie kommt an so vielen Stellen durch, dass man sich noch einmal die allerwichtigsten Grundlagen objektorientierter Systeme in Erinnerung rufen sollte. Neben den Charakterisierungen oben sind das noch die Bedeutung von Vererbung. Denn mit diesen Möglichkeiten geht die UML ständig um.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

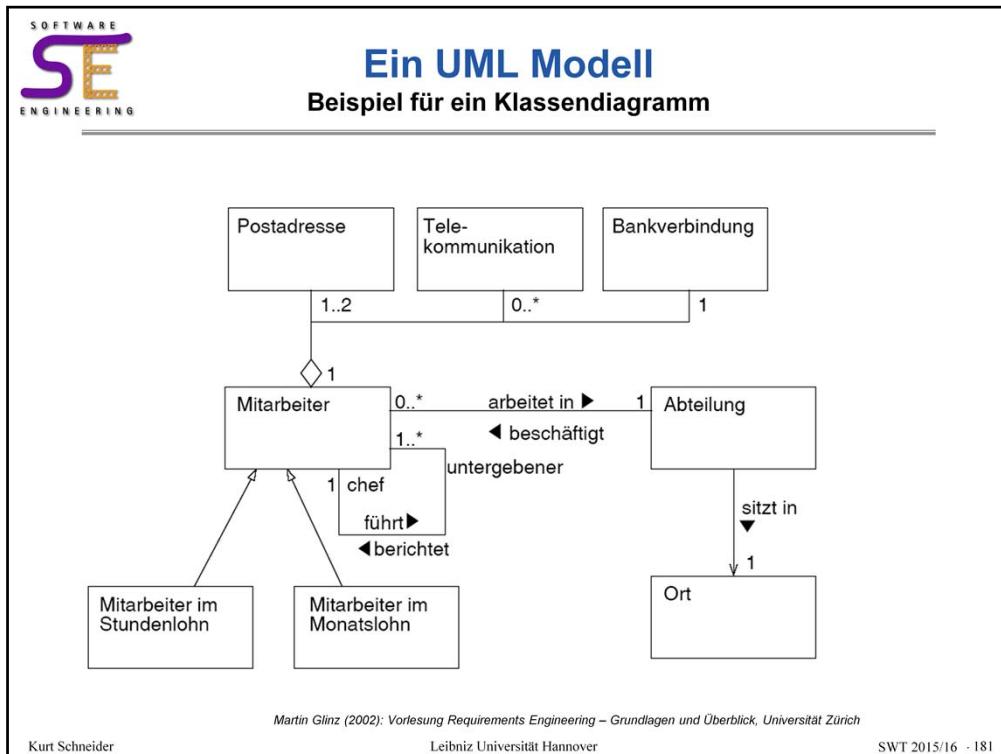


Verschiedene Diagrammtypen in UML.

Dabei jeweils die Sicht, die sie verkörpern (die durch sie repräsentierten relevanten Attribute).

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



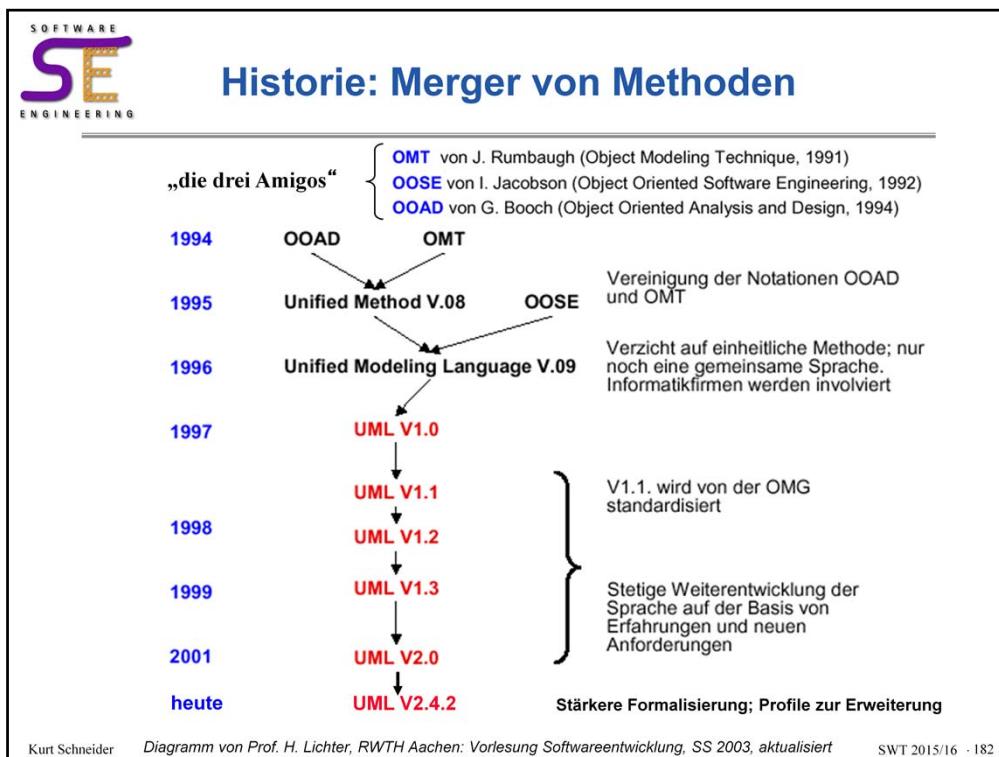
Dies ist ein UML-Klassendiagramm.

Gerade Klassendiagramme werden in sehr unterschiedlichem Detaillierungsgrad gezeichnet.

Hier sind die Assoziationen recht ausführlich, die Klassensymbole dagegen sehr simpel gezeichnet.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Objekt-orientierte Methoden sind anfangs ganz stark mit einigen wenigen Personen verbunden gewesen.

Drei wichtige Vertreter haben sich zusammengeschlossen und dadurch großen Einfluss auf die Entwicklung von objekt-orientierten Methoden genommen.

Allerdings hat die UML schon vor einigen Jahren den Anspruch aufgegeben, eine „Unified Method“ zu sein und hat sich auf den Notationsaspekt zurückgezogen.

Immerhin: durch die UML sind alternative Notationen fast ausgestorben. Da Standardisierung für einen (Software) Ingenieur erstrebenswert ist, verdankt das SE den „drei Amigos“ einiges.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



## UML-Diagramme bearbeiten

- Mein Tipp:
  - Malen Sie erst von Hand
  - Danach holen Sie sich ein Werkzeug, zum Beispiel:
    - StarUML (open source, kostenlos)
      - Download: <http://sourceforge.net/projects/staruml> (letzter Zugriff: 6.12.2014)
      - StarUML ist nicht hypermodern, aber leicht zu handhaben
      - Viele andere Werkzeuge sind für unsere Zwecke weniger geeignet
    - Auch gut: DIA Diagramm Editor
      - open source: <http://dia-installer.de/> (letzter Zugriff: 6.12.2014)
  - Wozu Sie StarUML oder DIA verwenden sollten
    - Nach einer Handskizze: Diagramme sauber abzeichnen
    - Speichern, Datei im Team verteilen und verwenden
    - Ändern und aktualisieren
  - Generieren, C# usw.?
    - Es geht noch viel mehr mit DIA, StarUML und ähnlichen Werkzeugen
    - Das brauchen Sie in SWT noch nicht
  - Welche UML-Variante gilt für die Klausur? Siehe: UML-Poster V2 von SE

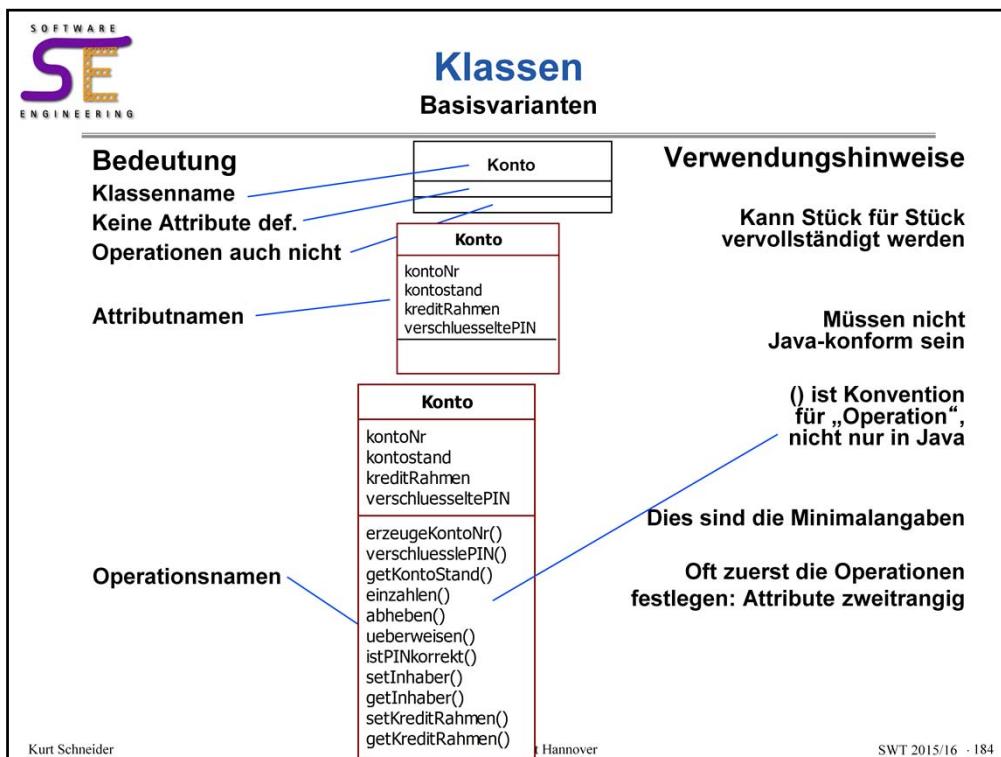
Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 · 183

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Hier sieht man ein Klassensymbol aus dem UML-Klassendiagramm in unterschiedlichem Detailgrad.

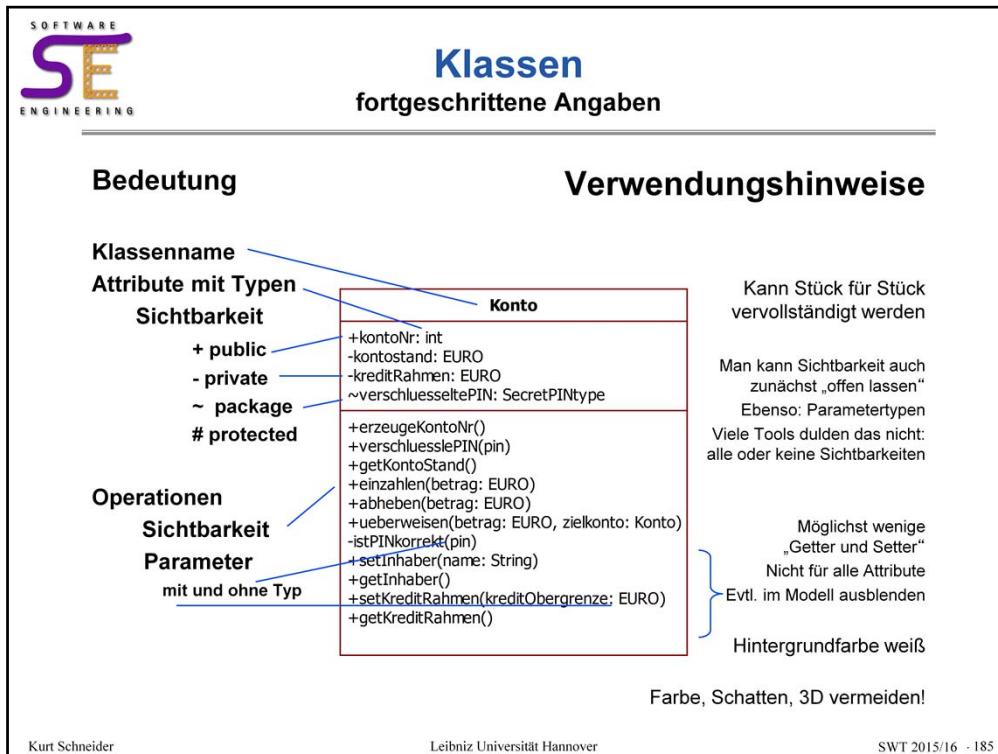
Beachten Sie, dass die () für Operationen vorgeschrieben sind. Sie bedeuten NICHT:

- Dass es sich um Java handelt
- Dass die Methode keine Parameter hat
- Dass die Methode keinen Rückgabewert liefert

Das kann so sein, muss aber nicht.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Hier ist noch mehr angegeben.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The diagram illustrates the relationship between UML modeling and code generation. It features a central UML class diagram for 'Konto' and two columns of code: C++ on the left and Java on the right. A legend at the top indicates that the purple background represents C++ and the yellow background represents Java.

**UML-Modell (Konto)**

Konto
kontoNr kontostand kreditRahmen verschluesseltePIN
erzeugeKontoNr() verschluesseltePIN() getKontoStand() einzahlen() abheben() ueberweisen() istPINkorrekt() setInhaber() getInhaber() setKreditRahmen() getKreditRahmen()

**C++ (Konto.h)**

```
#if !defined(_KONTO_H)
#define _KONTO_H
#include "Kunde.h"

class Konto {
public:
    int kontoNr;
    int erzeugeKontoNr();
    SecretPINtype verschluesseltePIN(int pin);
    EURO getKontoStand();
    void einzahlen(EURO betrag);
    EURO abheben(EURO betrag);
    void ueberweisen
        (EURO betrag, Konto zielkonto);
    void setInhaber(String name);
    String getInhaber();
    void setKreditRahmen
        (EURO kreditObergrenze);
    EURO getKreditRahmen();

private:
    EURO kontostand;
    EURO kreditRahmen;
    boolean istPINkorrekt(int pin);
};


```

**Java**

```
public class Konto {
    public int kontoNr;
    private EURO kontostand;
    private EURO kreditRahmen;
    public SecretPINtype verschluesseltePIN;

    public int erzeugeKontoNr() { }
    public SecretPINtype verschluesseltePIN
        (int pin) { }

    public EURO getKontoStand() { }
    public void einzahlen(EURO betrag) { }
    public EURO abheben(EURO betrag) { }

    public void ueberweisen
        (EURO betrag, Konto zielkonto) { }

    private boolean istPINkorrekt
        (Object pin) {}

    public void setInhaber(String name) { }
    public String getInhaber() { }
    public void setKreditRahmen
        (EURO kreditObergrenze) { }

    public EURO getKreditRahmen() { }
}
```

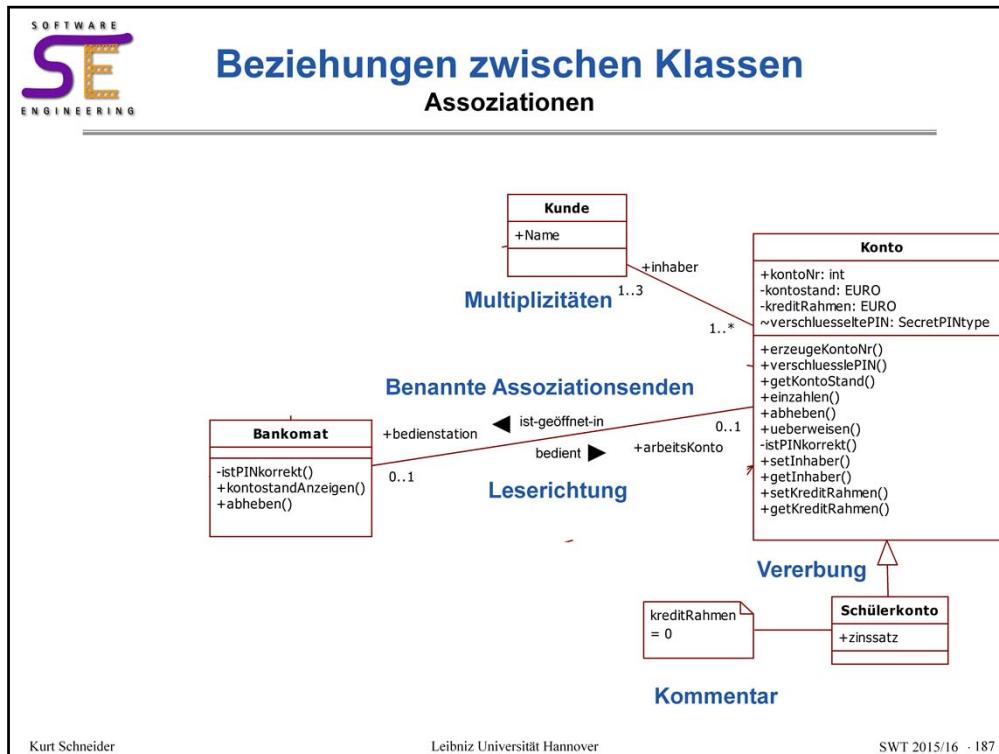
Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 · 186

Man kann mit demselben Klassensymbol verschiedene Programmiersprachen generieren.

Das Symbol „bedeutet“ verschiedenen Code in C++ und in Java

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Jetzt sind mehrere Klassensymbole verbunden.

Man sieht Leserichtungspfeile und Multiplizitäten (das sind praktisch Kardinalitäten von Assoziationen).

Ein Vererbungspfeil ist auch zu sehen.

Außerdem ein Kommentar. Diese sind in allen UML-Diagrammen erlaubt und sehen immer so aus.

Die gestrichelte Linie zeigt auf den kommentierten Modellteil.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

**Klassen und Objekte**

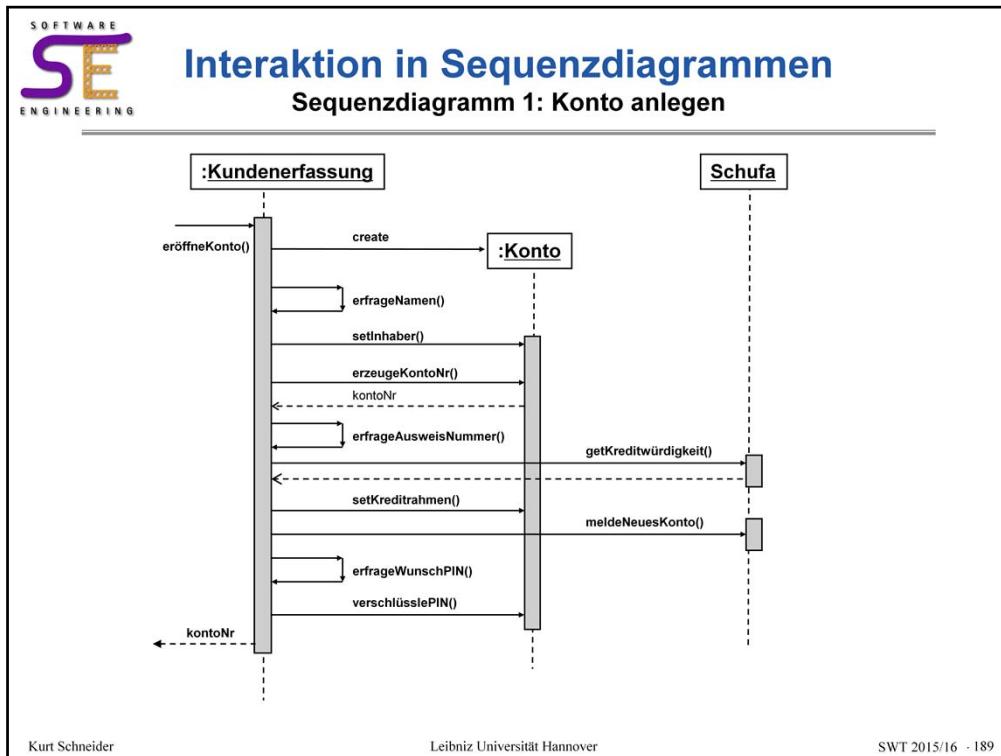
Klasse ganz einfach	Konto	:Konto	Anonym: Objekt hat keinen Namen
Klasse Mit mehr Angaben	<b>Konto</b>  kontoNr kontostand kreditRahmen verschluesseltePIN	<b>privat:Konto</b>  <b>vereinskonto:Konto</b>  <b>vereinskonto</b>	Objektname „privat“ und Klasse  Andere Instanz derselben Klasse  Variante: Ohne Nennung der Klasse

- Klassen und Objekte
  - Sehen ähnlich aus
  - Sollen aber auf den ersten Blick unterscheidbar sein
  - Daher gilt (zumindest in dieser Vorlesung):  
Namen von Objekten unterstreichen, von Klassen nicht

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 · 188

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

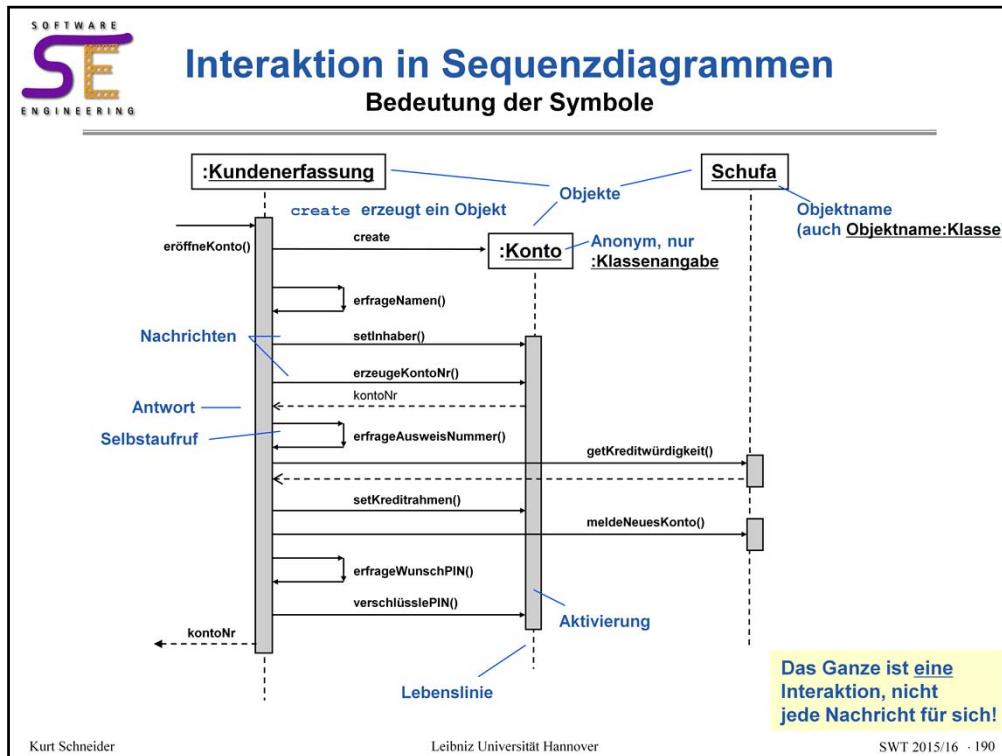
Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



In Sequenzdiagrammen sind Interaktionen zwischen Objekten dargestellt.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Die Objekte sind in den Rechtecken vermerkt (die Syntax beachten!).

Jedes Objekt hat eine Lebenslinie (gestrichelt), die angibt, wie lange das Objekt existiert.

Der graue Balken ist darin die Aktivitätsdauer des Objekts und zeigt, wie lange das Objekt an der modellierten Interaktion beteiligt ist.

Oben wird ein Objekt von der Klasse Konto erzeugt (Schlüsselwort new).

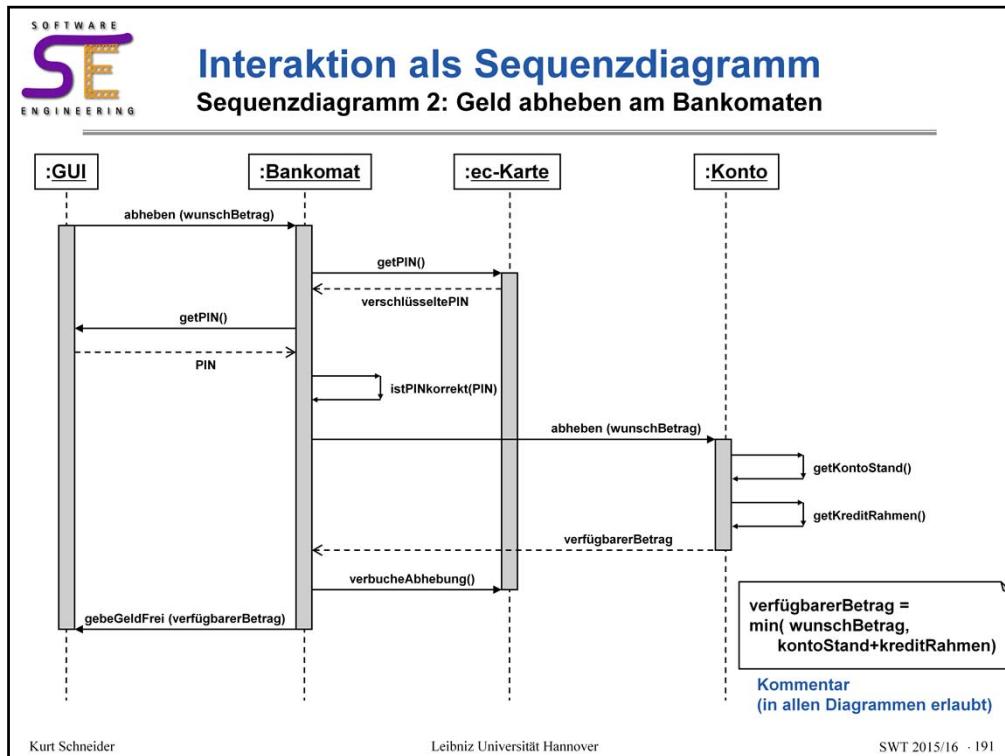
Die horizontalen Pfeile sind Nachrichten zwischen den Objekten. Die gestrichelten davon sind Antworten.

Man muss die Nachrichten bezeichnen und kann die Rückantworten benennen.

Gestartet wird die Interaktion durch eine Nachricht von außen.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



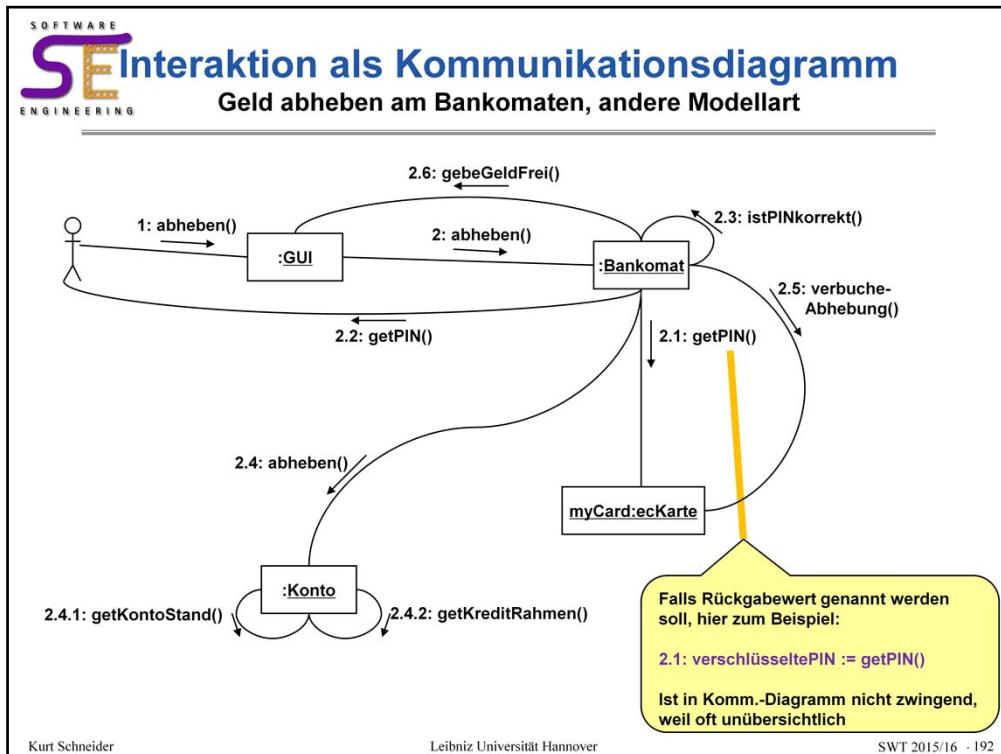
Ein weiteres Beispiel nach demselben Muster.

Der Kommentar ist zwar interessant, wird aber nicht formal interpretiert.

Hier könnte auch Freitext stehen.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



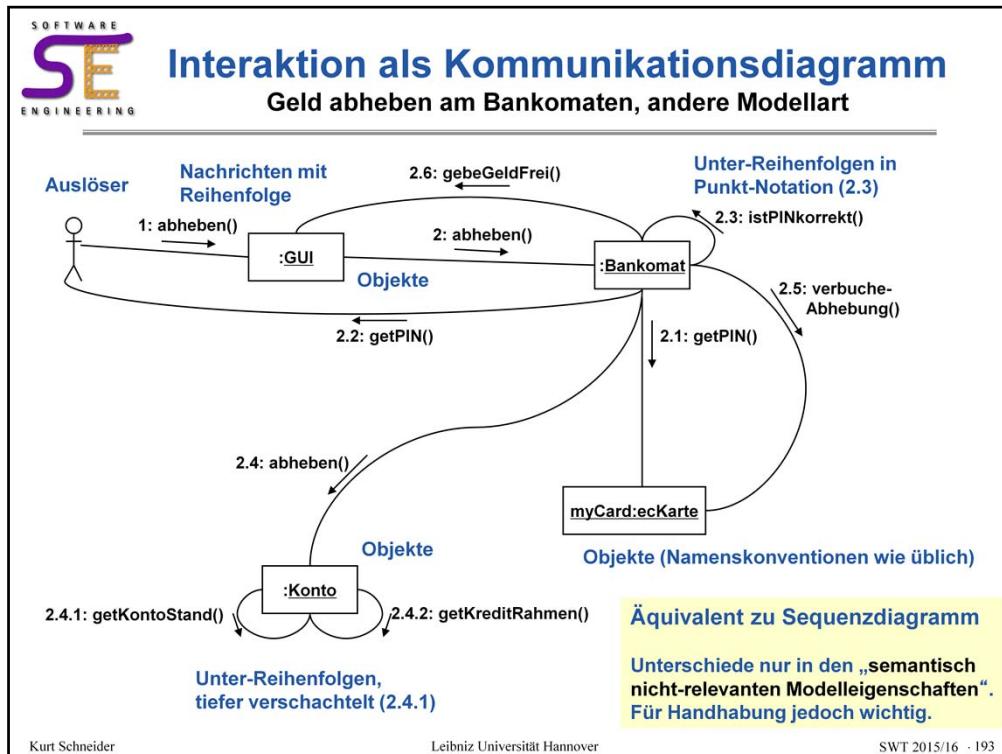
Man kann dieselbe Interaktion mit einer anderen UML-Diagrammart auch ganz anders modellieren, nämlich mit einem Kollaborationsdiagramm.

Wieder sieht man die Objekte, aber nun keine Lebens- oder Aktivitätszeiträume. Vielmehr werden Nachrichten durch Pfeile dargestellt.

Charakteristisch ist, dass die Nachrichten hierarchisch nummeriert sind, man also den Ablauf durch Verfolgen der Nummern rekonstruieren kann.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



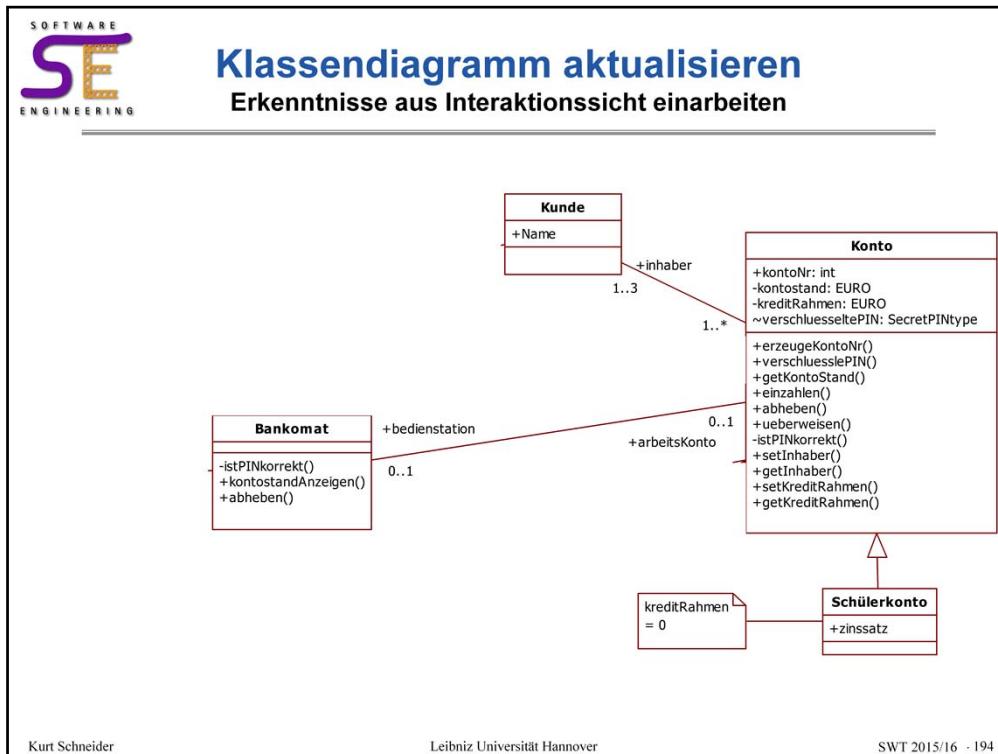
Man kann dieselbe Interaktion mit einer anderen UML-Diagrammart auch ganz anders modellieren, nämlich mit einem Kollaborationsdiagramm.

Wieder sieht man die Objekte, aber nun keine Lebens- oder Aktivitätszeiträume. Vielmehr werden Nachrichten durch Pfeile dargestellt.

Charakteristisch ist, dass die Nachrichten hierarchisch nummeriert sind, man also den Ablauf durch Verfolgen der Nummern rekonstruieren kann.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Oft wird man mit dem Klassendiagramm beginnen und dann die Interaktionen modellieren.

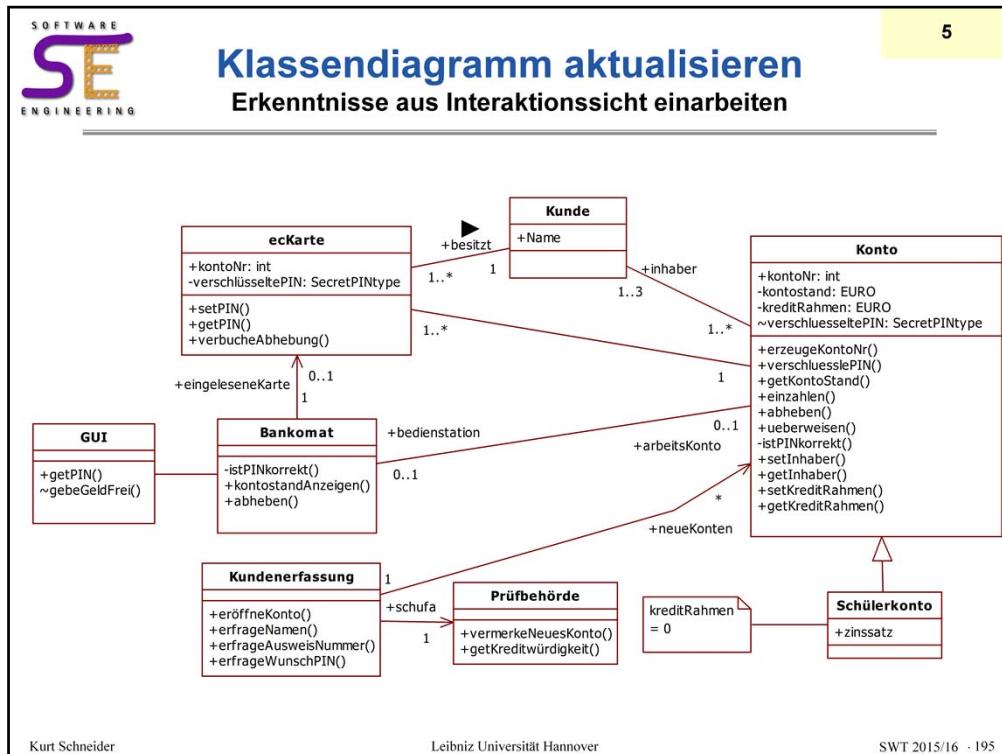
Nicht selten stellt man dabei fest, dass die Klassen andere Operationen oder Assoziationen haben sollten.

Das ist nicht schlimm. Man wird nun also das Klassendiagramm aktualisieren.

Am Schluss muss alles zusammenpassen: Eine Nachricht, die nicht als Operation einer Klasse im Klassendiagramm steht, kann nicht im Sequenzdiagramm verschickt werden.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

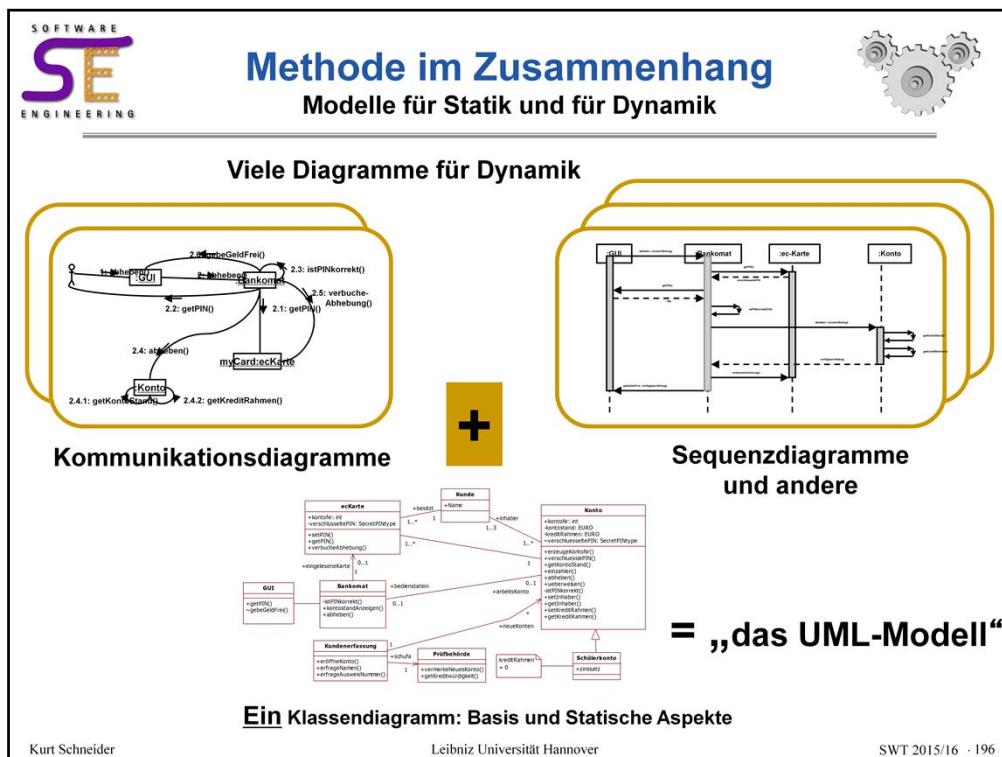
Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Hier hat sich doch einiges geändert.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Dieses Bild soll zeigen, wie aus vielen Diagrammen ein Programm wird.

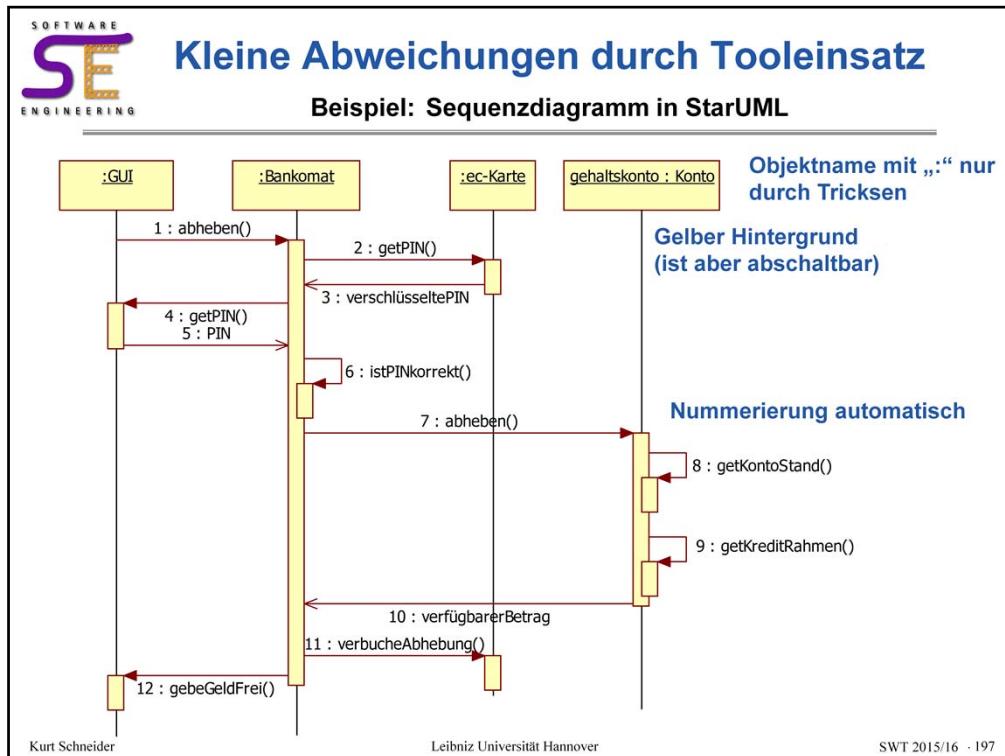
Man braucht (in der Regel: ein) Klassendiagramm, das die Attribute und Operationen zeigt, dazu die Aufrufmöglichkeiten. Welche Methoden gebraucht werden, kann man aus allen Interaktions- und Kollaborationsdiagrammen ablesen: Jede Methode, die dort irgendwo vorkommt, muss man auch im Klassendiagramm finden können.

Was diese Methoden tun, steht dann in den einzelnen Diagrammen, die Dynamik repräsentieren. Also kann man praktisch den Code aus den Dynamik-Modellen ableiten. Alles, was modelliert ist, muss man mit den angebotenen Methoden tun können.

Wenn man den Code generieren möchte, muss man allerdings die Modelle sehr genau beschreiben – genauer, als für Menschen, die die Diagramme lesen wollen.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

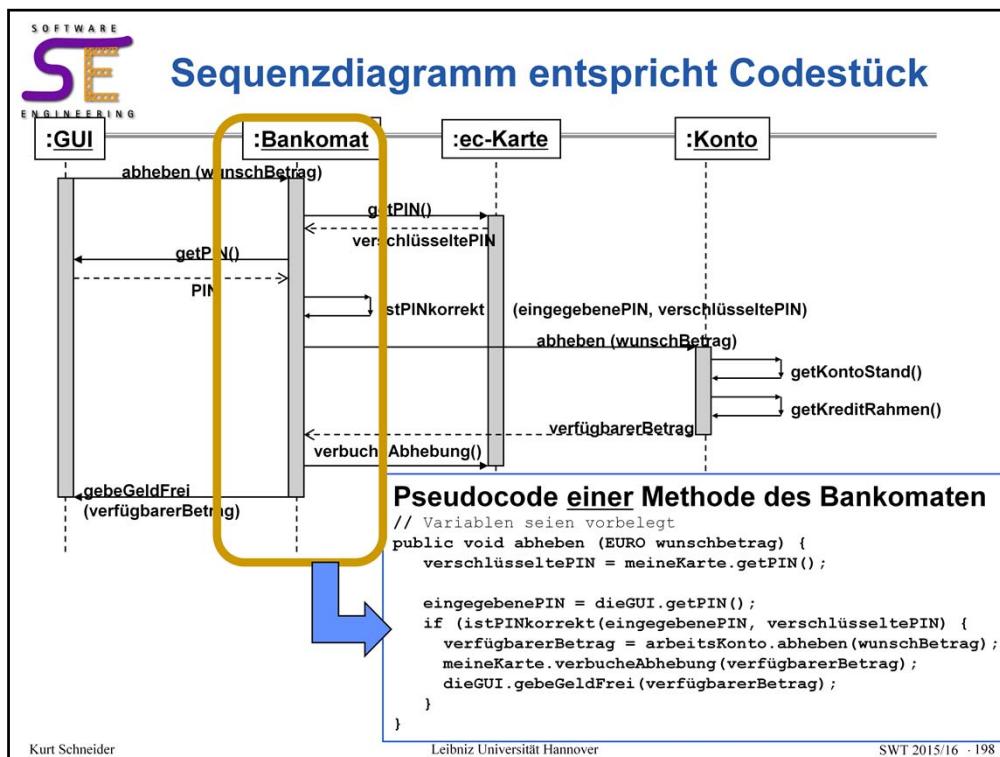


Jetzt noch einmal das bekannte Beispiel, allerdings mit dem empfohlenen (weil einfachen und kostenlosen) Werkzeug StarUML.

Einige Unterschiede sind angegeben und werden diskutiert.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

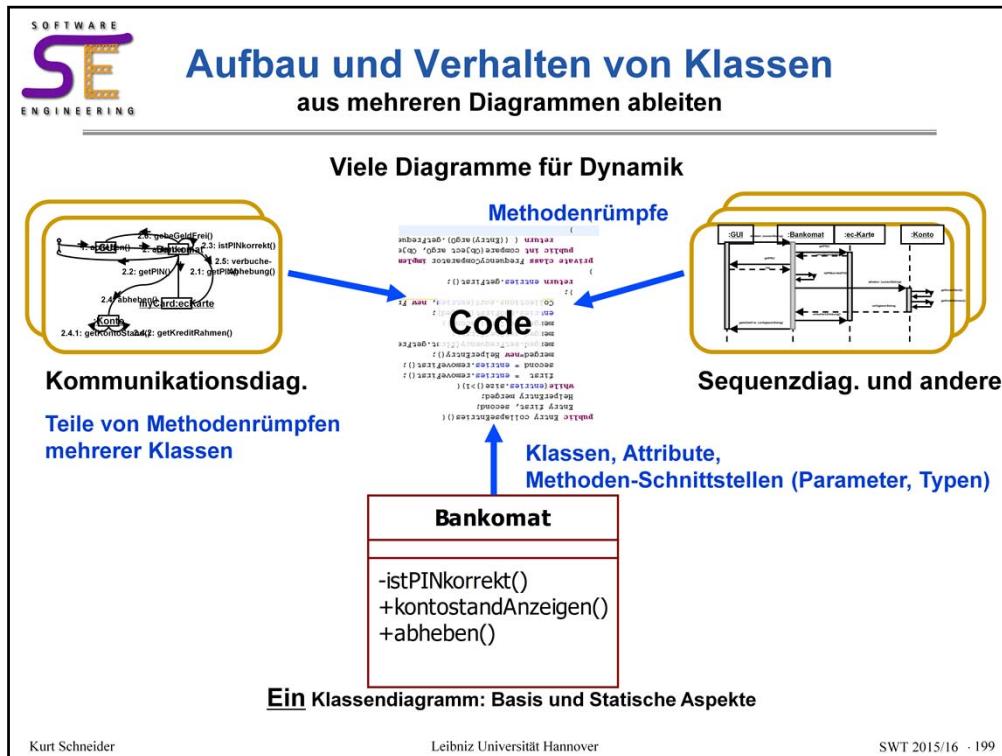


Grob gesprochen kann man aus den Modellen schon einen gewissen Teil des Codes generieren, aber hier noch nicht alles.

Aus einem Sequenzdiagramm lassen sich mehrere Codestücke „fast“ ableiten. Oben sieht man, wie nur für den Bankomaten die hier gezeigte Operation abheben umgesetzt wird. Einige Teile des Pseudocodes stecken aber nicht im Sequenzdiagramm und können daher auch nicht daraus generiert werden (z.B. „meineKarte“)

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Ohne hier auf Details einzugehen tragen also alle UML-Diagrammtypen dazu bei, den Code näher zu beschreiben.

Die Statik kommt von dem einen Klassendiagramm, viele Aspekte der Dynamik kommen aus den Interaktionsdiagrammen, wie vorne angedeutet.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



## Konsistenz von Diagrammen

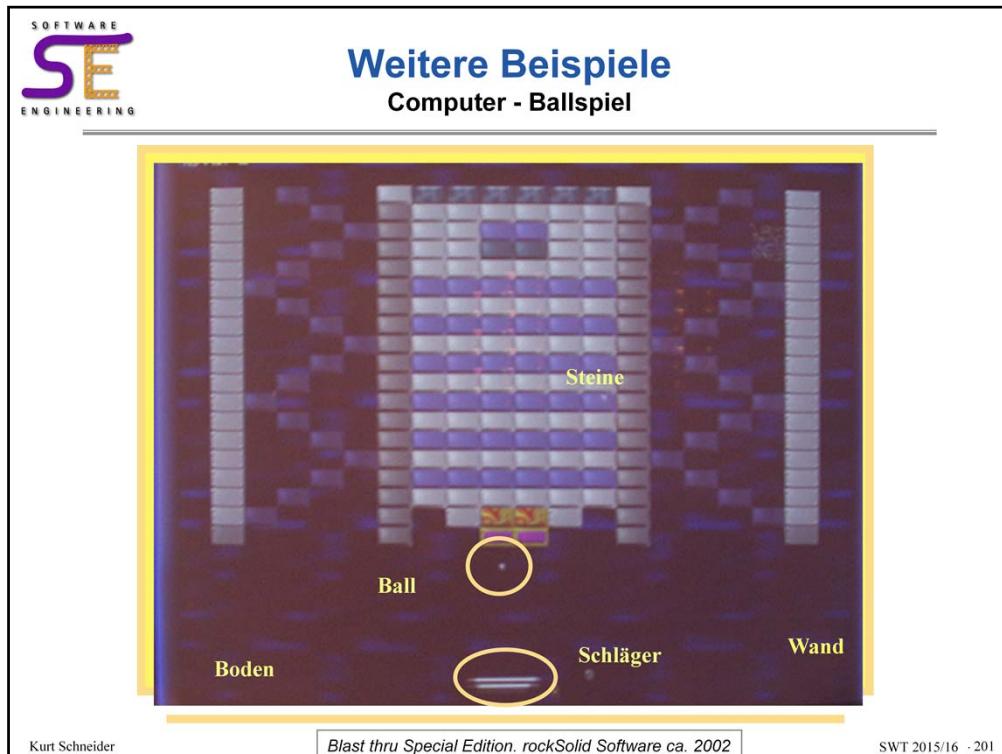
- Ein UML-Modell ist auf mehrere UML-Diagramme verteilt
- In der Regel verschiedene relevante Aspekte betont
- Überlappungen müssen konsistent sein
  - Objekte passen zu Klassen aus dem Klassendiagramm
  - Sichtbarkeiten, Parameter und Typen sind konsistent dargestellt
  - Methodennamen in Sequenzen passen zu Klassendiagramm
  - Use Case Modelle (kommen noch) passen zu Interaktionspersp.
  - Verschiedene Interaktionsdiagramme widersprechen sich nicht
- Zweck aller Diagramme in einem Modell
  - Konsistente Darstellung aller relevanten, interessanten Aspekte
  - Genau genug für Programmierung
  - Möglicherweise sogar für (teilweise oder vollständ.) Generierung

Kurt Schneider    Leibniz Universität Hannover    SWT 2015/16 · 200

Damit alles zusammenpasst, müssen Konsistenzregeln beachtet werden.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Objekte zu identifizieren geht erst einmal ganz einfach.

Wichtig für das Spiel (allgemein: die Anwendung des Kunden) sind die Dinge, die in einer kurzen Beschreibung des Kunden von dem, was er möchte, vorkommen.

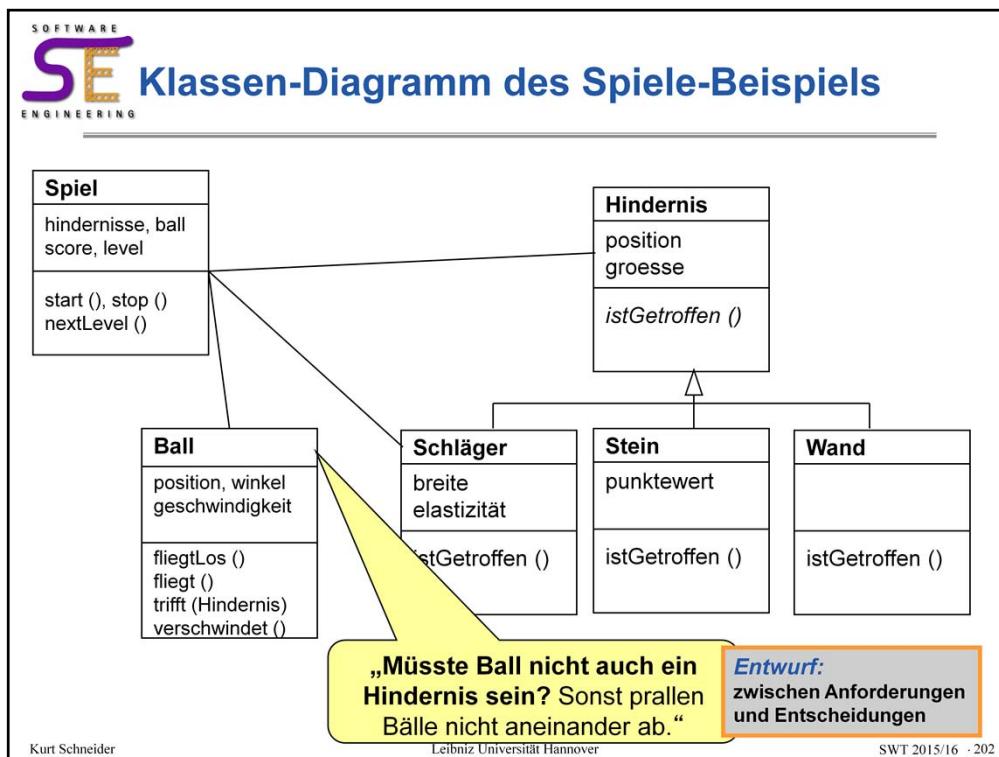
Wand und Boden sind dagegen nicht so offensichtlich und werden in aller Regel erst auf Nachfrage als Objekte genannt.

Sowohl Wand als auch Boden haben in diesem Spiel sowohl einen Zustand (Position) als auch ein Verhalten (Wand lässt Ball zurückprallen, Boden verschluckt ihn). Damit sind sie richtige Objekte.

Übrigens verhalten sich nicht alle Steine gleich; hier könnte man also von verschiedenartigen Objekten sprechen, die andererseits wieder viel gemeinsam haben (z.B. verschwinden sie, wenn sie getroffen werden und sie haben rechteckige Form).

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Die Objekte der Situation sind die einzelnen Teile.

Bei den Steinen ist offensichtlich, dass (gleichfarbige) Steine sich auch gleich verhalten. Aber natürlich nur, wenn sie den gleichen Reizen/Interaktionen ausgesetzt sind: Der eine (blaue) Stein wird getroffen und verschwindet, der andere (blaue) Stein daneben wird nicht getroffen und verschwindet nicht. Die Steine sind also Individuen mit gleich-artigem Verhalten aber unterschiedlichen Zuständen.

In der Objektorientierung geht man nun von gleich-artigen Objekten zu einer Klasse von Objekten über (ähnlich der Klassenbildung in der Mathematik, in der man alle Elemente in eine Klasse steckt, die sich bezüglich einer definierten Äquivalenzrelation nicht unterscheiden). Zu einer Klasse gehört: Gleich-artigkeit des Verhaltens und Gleich-strukturiertheit der Zustände (aber nicht: gleiche Werte der Zustandsvariablen).

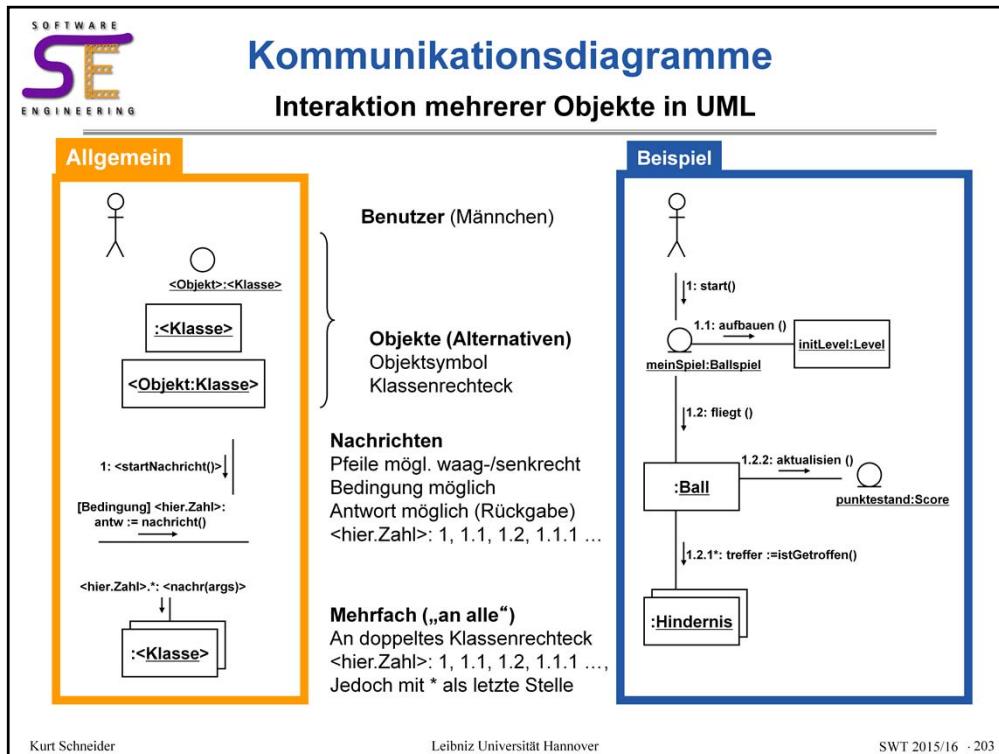
Ein Pfeil mit leerer Spitze zeigt von einer Unterklassie auf ihre Oberklasse. Damit kann man ausdrücken: jeder Stein ist ein Hindernis (jede Wand auch), aber nicht jedes Hindernis ist ein Stein. Alles, was man allgemein über Hindernisse sagen kann (sie haben eine Position und Größe und können getroffen werden), kann man dann auch über alle Unterklassen sagen. Das nennt man Vererbung.

Zusätzlich hat jede spezielle Unterklassie noch ein paar spezifische Eigenheiten – zum Beispiel einen Punktwert und eine andere Reaktion auf das Getroffen-Werden (das Verhalten bzw. die Operation steht hier noch einmal: die spezifische Ausprägung der Unterklassie „überschreibt“, d.h. überstimmt, die geerbte der Oberklassie.).

Wichtig: Das *Spiel* ist auch ein Objekt, und jedes Objekt hat eine Klasse.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

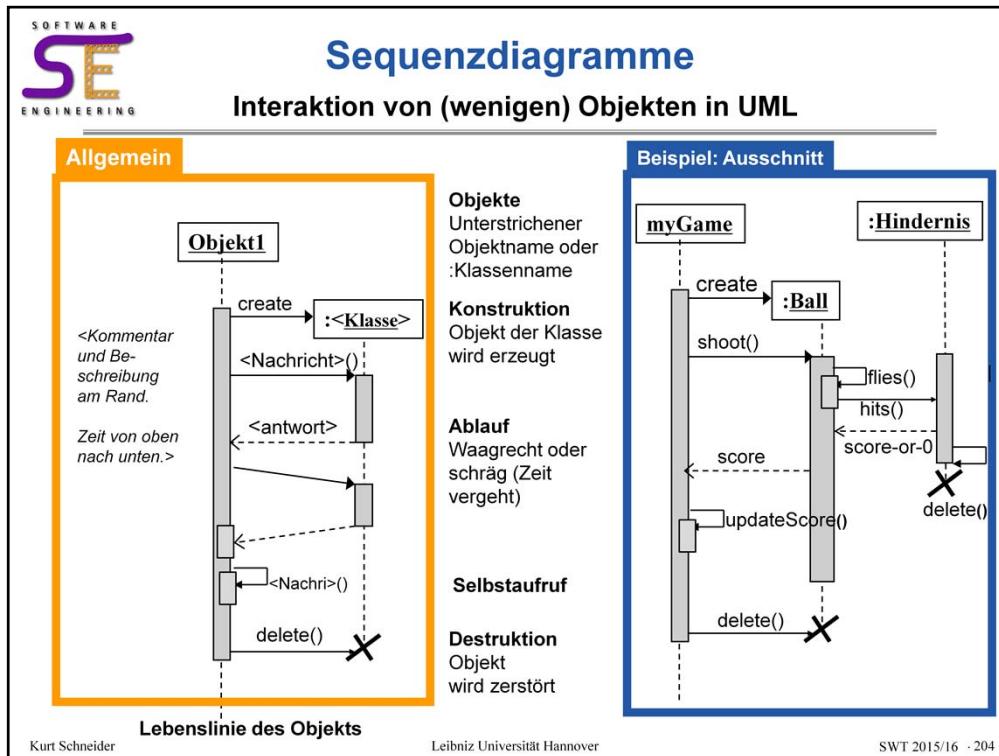


Das Kollaborationsdiagramm zeigt, wie viele miteinander agierende Objekte beschrieben werden. Zentral ist die hierarchische Nummerierung, an der man sich entlangtasten kann.

Etwas verwirrend mag sein, dass es so viele verschiedene Formen gibt, ein Objekt zu zeichnen.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



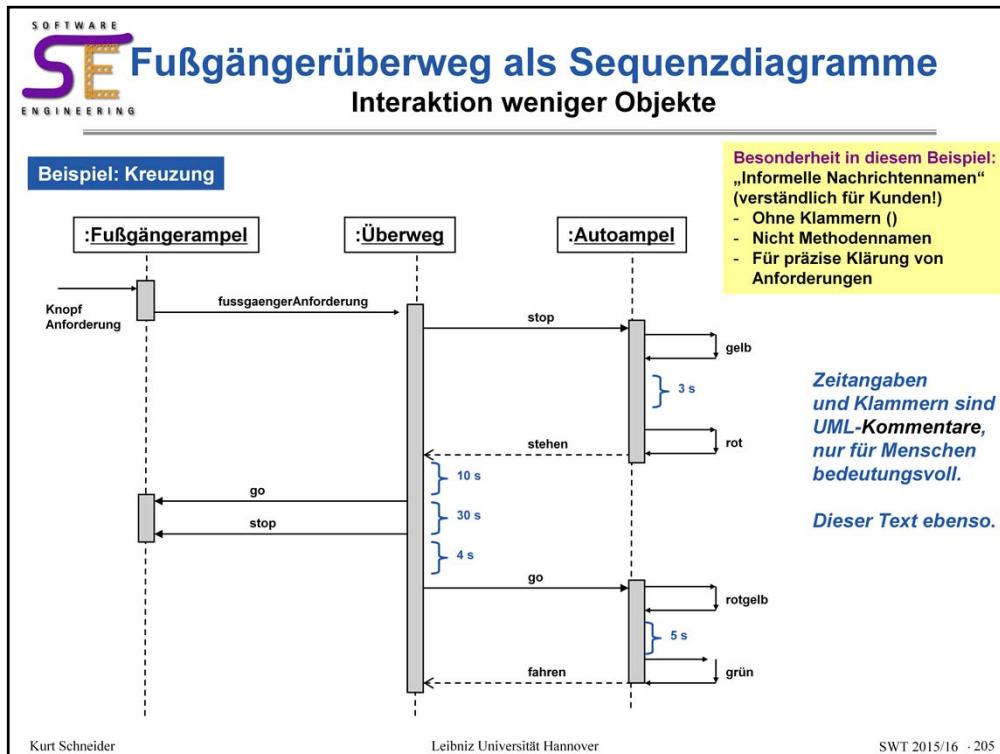
Das Hin und Her in Sequenzdiagrammen ist besonders gut geeignet, um eine intensive Interaktion zwischen relativ wenigen Objekten zu beschreiben. Solange deren Lebenslinie läuft, kann das Objekt angesprochen werden. Wenn es aktiv ist, sieht man die verbreiterte Lebensline.

Was zuvor noch nicht eingeführt wurde: Das Zerstören von Objekten mit delete. Damit endet deren Lebenslinie.

Ferner: Selbstaufrufe, also Nachrichten eines Objekts an sich selbst. Durchaus sinnvoll!

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Dies hat einige Vorteile:

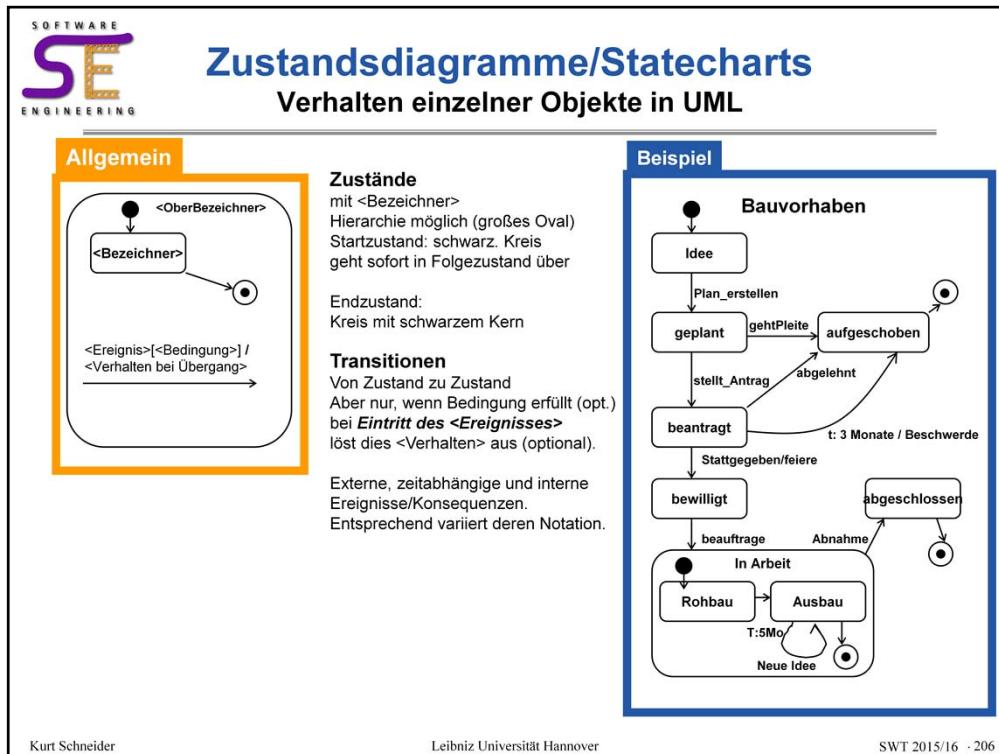
- Man kann chronologisch verfolgen, wer was tut.
- Man sieht, dass es keine Racing conditions gibt
- Zuständigkeit für zwischenzustände ist klarer
- Die Methoden rot, gelb usw. sind offensichtlich

- WICHTIG: Hier kann man auch bei grüner Fußgängerampel anfordern, das ging vorne nicht.

-Nachteile: Die Zeitbehaftung ist in Zustandsdiagrammen besser ausdrückbar.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Kurt Schneider

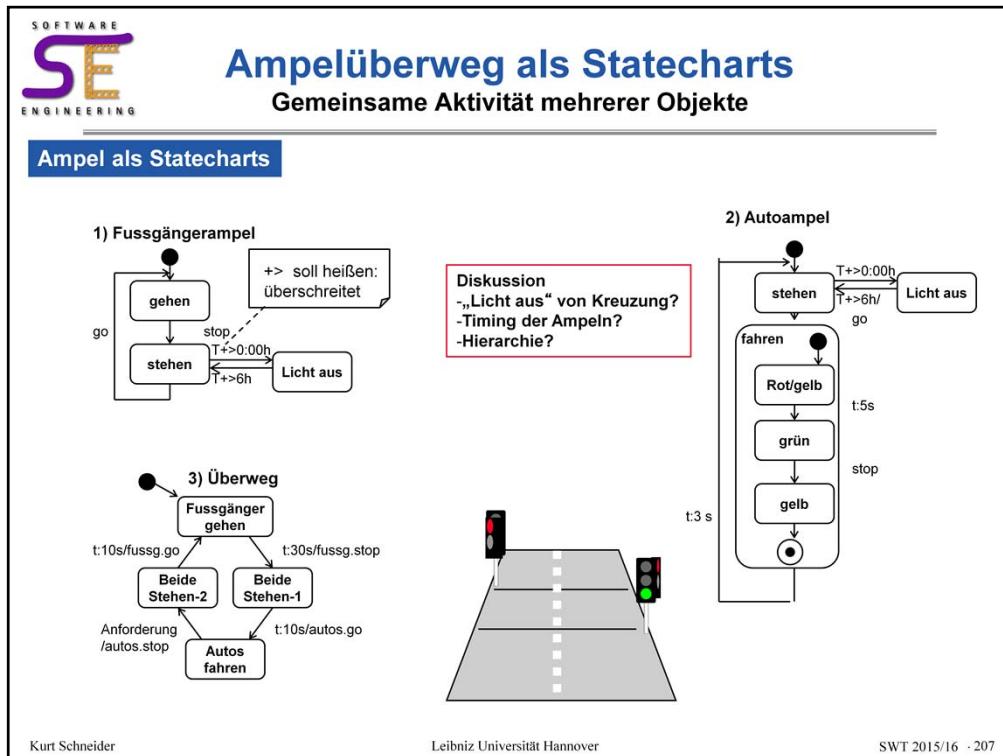
Leibniz Universität Hannover

SWT 2015/16 · 206

Die dritte Notation sind Zustandsdiagramme. Zustände können verschachtelt (hierarchisch) sein. An jedem Punkt bzw. in jedem Zustand können mehrere beschriftete Kanten abgehen. Sie stehen für die Bedingungen, unter denen der eine oder andere Folgezustand angenommen wird.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



An einem etwas umfangreicheren Beispiel sieht man manche Schwierigkeiten:

Während die Fußgängerampel mit zwei Zuständen (wie Lampen) auskommt, braucht die Auto-Ampel mehr. Schließlich brennen auch zwei Lampen gleichzeitig.

Im Sinne der Ampelkreuzung müssen die beiden Sorten von Ampeln noch koordiniert werden, dafür steht das Statechart links unten.

Aber ganz so einfach ist das nicht, denn es kann zu ungünstigen Bedingungen kommen, wenn der übergeordnete Kreuzungs-automat mit den untergeordneten Ampel-Automaten schlechte Zeitverhältnisse und timing conditions hat.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The diagram is titled "Vergleich der drei Interaktionsdiagramme" and features a logo for "SOFTWARE ENGINEERING" with "SE" in large letters. It lists three types of interaction diagrams:

- Sequenzdiagramme**
  - Komplizierte Abläufe zwischen wenigen Objekten
  - Exakte Abfolge auf einen Blick sichtbar
  - Komplizierte, reiche Notation in UML
- Kommunikationsdiagramme**
  - Zusammenarbeit mehrerer Objekte
  - Abfolge nicht auf einen Blick sichtbar
  - Relativ einfache Notation in UML
- Zustandsdiagramme/Statecharts**
  - Basis: Hierarchische Zustandsautomaten
    - (siehe Statecharts von Harel)
  - Kompliziertes Innenleben einer Klasse, eines Use Case usw.
  - Zusammenarbeit mit anderen nicht so zentral

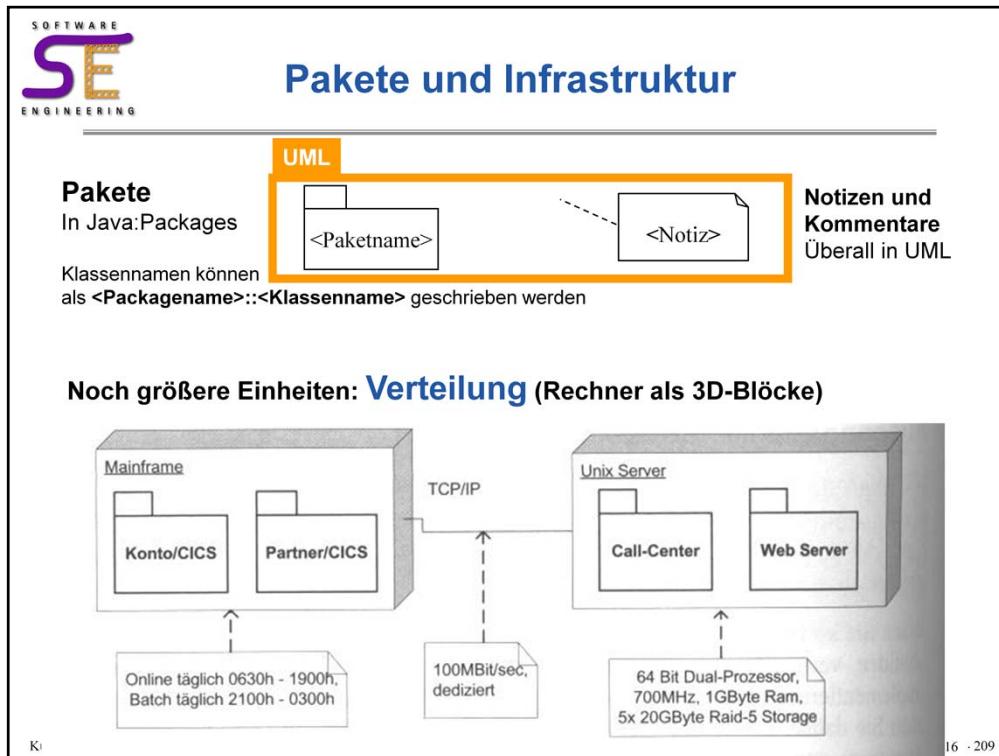
Annotations in yellow boxes:

- Oberbegriff: Interaktionsdiag.**  
Für je ein zentrales Szenario in Use Cases  
Nächster Schritt: Code
- Für jedes UML-Element (Domain Model, Klasse, Use Case) einsetzbar.  
Ziel:  
Genaues Innenleben klären.**

Page footer: Kurt Schneider, Leibniz Universität Hannover, SWT 2015/16 · 208

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



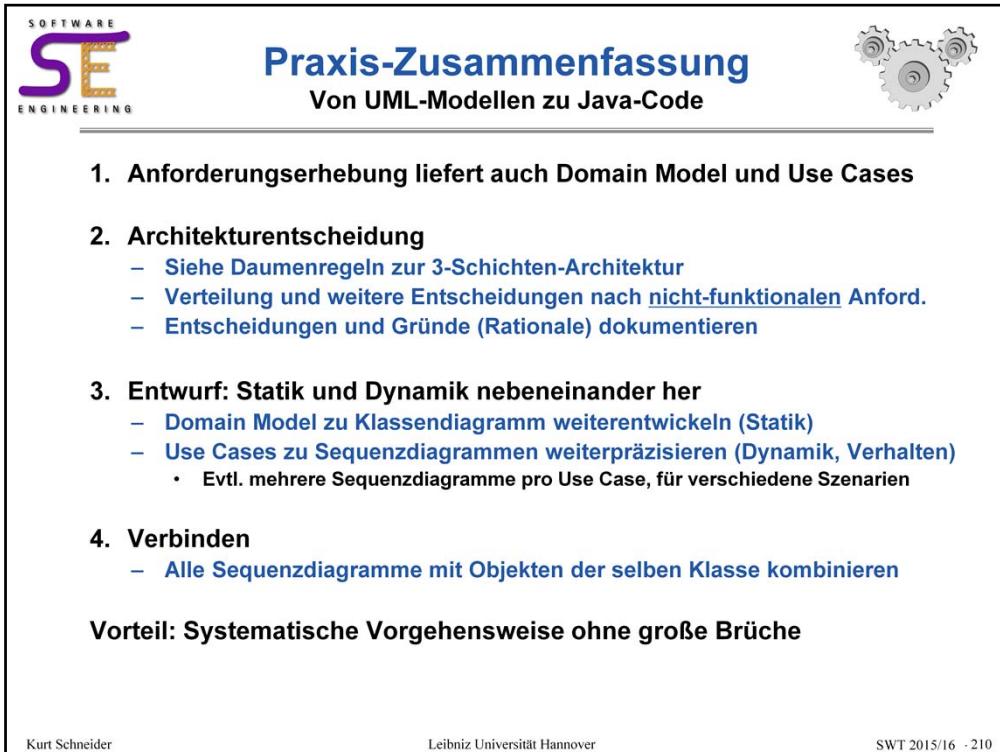
Ein Beispiel für die Infrastruktursicht.

Hier sieht man sehr große, grobe Bestandteile und etliche Kommentare.

Die Kommentare machen einen großen Teil des Werts eines solchen Diagramms aus.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



The slide template features a header with the logo 'SOFTWARE ENGINEERING' and the text 'Praxis-Zusammenfassung Von UML-Modellen zu Java-Code'. It includes a decorative gear icon in the top right corner. The main content area contains a numbered list of steps: 1. Anforderungserhebung liefert auch Domain Model und Use Cases; 2. Architekturentscheidung (with sub-points about 3-layer architecture, distribution, and rationales); 3. Entwurf: Statik und Dynamik nebeneinander her (with sub-points about Domain Model, Class Diagrams, Use Cases, Sequence Diagrams, and scenarios); 4. Verbinden (with a sub-point about combining objects from the same class). A bolded note 'Vorteil: Systematische Vorgehensweise ohne große Brüche' is placed below the list. The footer contains the names 'Kurt Schneider', 'Leibniz Universität Hannover', and 'SWT 2015/16 · 210'.

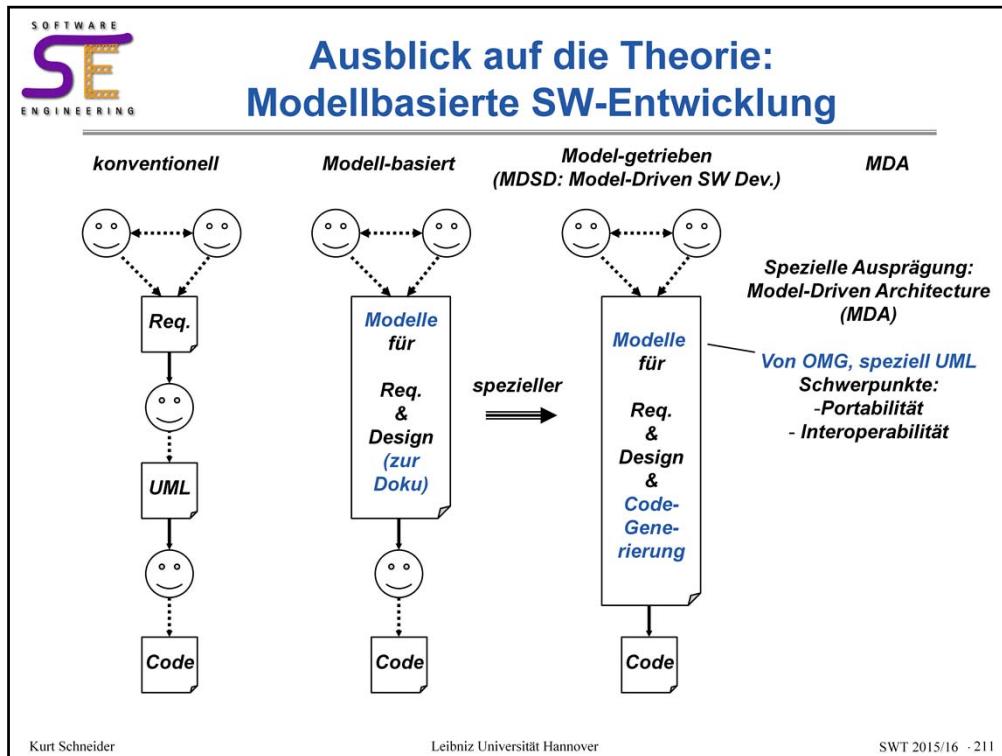
1. Anforderungserhebung liefert auch Domain Model und Use Cases
2. Architekturentscheidung
  - Siehe Daumenregeln zur 3-Schichten-Architektur
  - Verteilung und weitere Entscheidungen nach nicht-funktionalen Anford.
  - Entscheidungen und Gründe (Rationale) dokumentieren
3. Entwurf: Statik und Dynamik nebeneinander her
  - Domain Model zu Klassendiagramm weiterentwickeln (Statik)
  - Use Cases zu Sequenzdiagrammen weiterpräzisieren (Dynamik, Verhalten)
    - Evtl. mehrere Sequenzdiagramme pro Use Case, für verschiedene Szenarien
4. Verbinden
  - Alle Sequenzdiagramme mit Objekten der selben Klasse kombinieren

**Vorteil:** Systematische Vorgehensweise ohne große Brüche

Wie immer: Zusammenfassung und pragmatische Vereinfachung: Wie geht man in einem Projekt vor?

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Man kann das Generieren aus Modellen noch viel weiter treiben und versuchen, so weit wie möglich auf Programmieren von Hand zu verzichten.

Das ist Gegenstand der Master-Vorlesungen MEM und EKSE.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

## Softwaretechnik

### Inhalt von Kapitel 5



#### **Systematische Softwareentwicklung**

##### **5. Entwürfe notieren in UML – Modelle im Software Engineering**

- Historie der UML
- Objektorientierung: Konzepte und Denkweise
- Klassendiagramme
- Sequenz- und Kollaborationsdiagramme
- Information Hiding

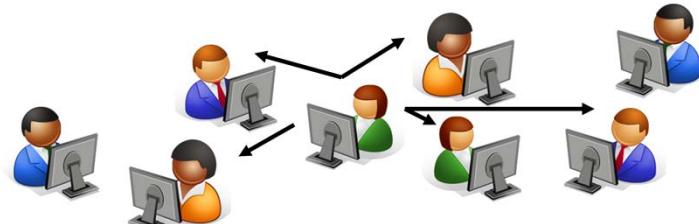
Leibniz Universität Hannover

SWT 2015/16 · 212

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

**SIE ARBEITEN IM TEAM**  
Stellen Sie sich vor ...



- Sie haben den Huffmann-Code programmiert.
- Nun soll ihn das ganze Team benutzen.
- Was geben Sie den Kollegen, damit sie das tun können?

Kurt Schneider                                    Leibniz Universität Hannover    SWT 2015/16 · 213

Hier arbeiten wir uns auf die Frage hin: Wieso möchte man möglichst WENIG wissen (müssen), wenn man ein fremdes Programmstück verwenden soll?

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The screenshot shows a software interface for generating a Huffman code. At the top, there's a logo for "SOFTWARE ENGINEERING" with "SE" in large letters. Below it, the title "Was müssen die anderen wissen? Beispiel Huffmann-Code-Programm" is displayed. The interface features several windows:

- A central window shows a Huffman tree structure where nodes merge based on their frequencies.
- To the left, a table lists characters A through F with their corresponding frequencies.
- Below the tree, a table titled "Huffmann-Code" shows the binary codes assigned to each character.
- A code editor window on the right contains Java code for generating the Huffman tree and code book.
- At the bottom, there are tabs for "All Classes", "CodeEntry", "Entry", "HelperEntry", "HuffmanEntry", "HuffmanCode", and "UsingHuffmanCodes".

Kurt Schneider

Leibniz Universität Hannover

SWT 2015/16 · 214

Viele Informationen, viele Entscheidungen: Die Datenstruktur, der Algorithmus, der genaue Code. Das will man alles nicht wissen.

Solange man genau weiß, wie man sich einen Huffmann-Code erzeugen lassen kann.

Vielleicht will man sogar das nicht wissen, sondern nur: Wenn ich in dieses Programm mit „addChar“ meine Zeichen einspeise, dann kriege ich mit „printCodeBook“ die Tabelle der Codes ausgegeben. Wie das geht, ist mir egal. Vielleicht versteckt man es vor mir (information hiding), aber auch wenn nicht: ich möchte mich damit gar nicht belasten. Hauptsächlich, das Ding funktioniert, wie in der Dokumentation beschrieben.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The diagram shows five people at computer terminals. Red arrows point from the left side towards the right side, indicating a shift in perspective or focus from general details to specific requirements.

**Perspektivenwechsel**  
Was möchten Sie in dieser Situation wissen (müssen)?

„Oh, es gibt jetzt einen effizienteren Algorithmus! Verwenden Sie doch den! Wir haben ein Modul gekauft. Es ist aber sehr kompliziert; 12 Seiten Mathe – wenn Sie das nicht verstanden haben, können Sie das Programm kaum richtig einsetzen. Aber wir haben auch die ganze Entwicklerdoku hier – drei Ordner! Super, oder?“

Wollen Sie das?

Oder eher das:

BetterCode
+addChar(char, int)
+printCodeBook()

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 · 215

Also möchte man eben nur die rechten Sachen wissen, die Schnittstelle.  
Nicht all die Details links.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The slide features a logo for "SOFTWARE ENGINEERING" with "SE" in large letters. The title "Also: Was müssen die anderen wissen?" is displayed in blue. Below it, the subtitle "Beispiel Huffmann-Code-Programm" is shown. A central diagram illustrates a team of five people working on computers, connected by arrows indicating communication or collaboration. On the left, a text box contains the German phrase "Das meiste verstecken wir! Zum Glück." (Most of it we hide! Fortunately). To the right, a code editor shows Java code for a "HuffmanCode" class. The code includes methods for adding characters and printing the codebook. A "Method Summary" table provides a detailed overview of these methods. The bottom of the slide includes navigation information: "Kurt Schneider", "Leibniz Universität Hannover", and "SWT 2015/16 · 216".

Method Summary	
void	<a href="#">addChar(char newChar, int frequency)</a> Include a new character.
void	<a href="#">printCodeBook()</a> Derive Huffmann code and print codebook for it (to standard out)

Die Schnittstelle als UML-Klasse (Rechteck, UML kommt später) und als html, erzeugt aus JavaDoc (unten).

Den Hintergrund wollen wir nicht sehen.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



## Entwurfsprinzip „Information Hiding“

- Von David Parnas, 1972
- Absolute Grundlage des SE!
  - Taucht immer wieder auf
  - Von Implementierung bis Doku.
- Idee: Jede Einheit gibt nur das Nötigste preis, um sie zu nutzen
  - Das ist die Schnittstelle mit Dokumentation
  - Wie sie funktioniert, ist dagegen versteckt (hidden)
    - Ändert sich etwas Internes, merkt man es außen nicht
    - Da vor ihm/ihr versteckt, braucht Nutzer keine Selbstdisziplin
    - „Jede Einheit versteckt (min.) eine Entwurfsentscheidung“ Parnas



Kurt Schneider Leibniz Universität Hannover SWT 2015/16 · 217

Information Hiding ist eines der absoluten Grundprinzipien des Software Engineering. Das müssen Sie in mehreren Spielarten kennen und verstanden haben.

Auch den Namen des „Erfinders“ David Parnas müssen Sie kennen. Einer der wenigen Namen in der Informatik, der wirklich berühmt ist. Zu Recht.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The slide has a header 'Beispiel/Szenario' and 'Vereinfachtes Telefonbuch'. In the top right corner, there is a blue box labeled 'Beispiel Interfaces'. On the left, there is a logo for 'SOFTWARE ENGINEERING' with the letters 'SE' in a stylized font. To the right of the title, there are two icons: a yellow smiley face with 'A' below it and an orange smiley face with 'B' below it. The main content is a bulleted list:

- A schreibt Customer Management System (CMS)
  - Will darin ein vereinfachtes Telefonbuch nutzen
  - Beauftragt B
- B macht sich Anforderungen klar
  - (Name, Telefonnummer)-Einträge verwalten
  - Suche nach richtigem Eintrag unterstützen
  - Löschen, Initialisieren
- Alles Kompliziertere lassen wir in diesem Beispiel weg!
  - Namen seien eindeutig, null werde abgefangen usw.
- Ziel: Sinn von Information Hiding mit Interfaces zeigen

At the bottom of the slide, there are three small text elements: 'Kurt Schneider', 'Leibniz Universität Hannover', and 'SWT 2015/16 · 218'.

Dieses Beispiel soll illustrieren und dabei motivieren, wieso es gut ist, Interna nicht wissen zu müssen.

Besser ist es, mit Schnittstellen zu arbeiten – und zwar für Aufrufende ebenso wie für Anbietende.

Gezeigt an einem Szenario.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The slide is titled "Erster Versuch: B verwendet Array". It features a logo for "SOFTWARE ENGINEERING" with "SE" in large letters. A sidebar note says "Einfach Schnell fertig". The main content area contains a code snippet for the "PhoneTable" interface:

```
PhoneTable(int size)
setEntry(int id, String name, String number)
getNumber(String name): String
getID(String name): int
delete(id)
```

Below this is another code snippet:

```
CMS verwendet PhoneTable
public void organize() {
    PhoneTable ph = new PhoneTable(7);
    ...
    ph.setEntry(3, "May", "089/4567");
    nr = ph.getNumber("Köhler");
    ...
    ph.delete(ph.getID("Huber"));
    ...
}
```

To the right, it says "Als simples Array implementiert" and shows a table:

1	Müller	0511/97865
2	Huber	0241/12-4711
3		
4	Schneider	762-19666
5	Schmidt	001-(303) 492
6		
7	Köhler	030/762-123

At the bottom, there are credits: "Kurt Schneider", "Leibniz Universität Hannover", and "SWT 2015/16 · 219". A blue bar at the top right says "Beispiel Interfaces".

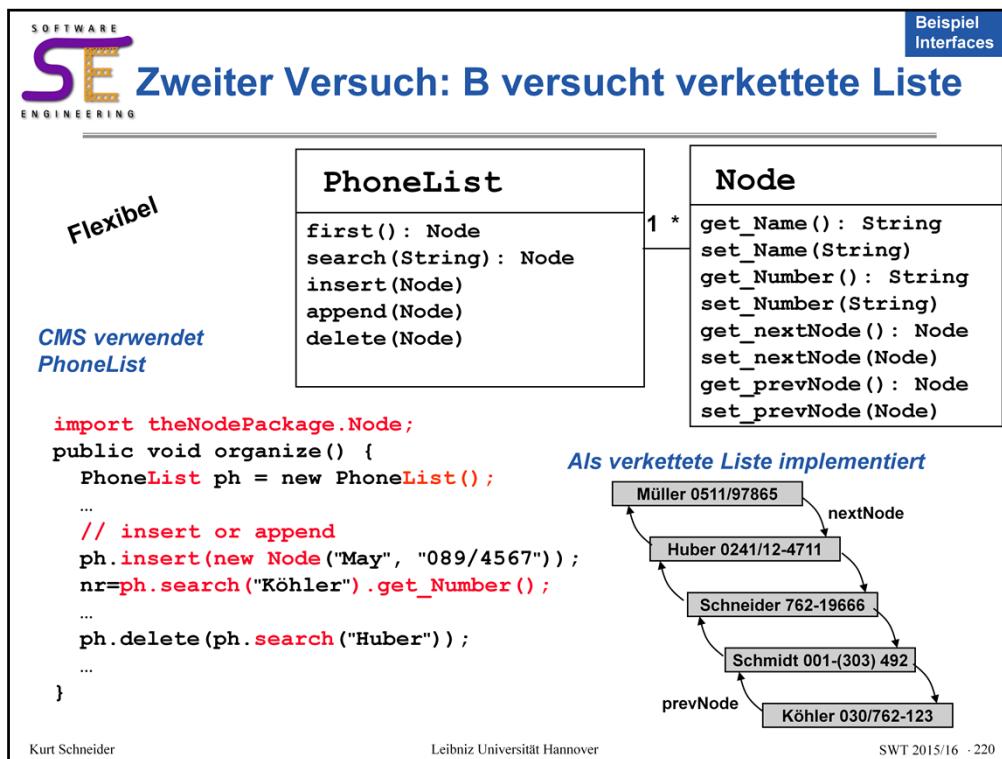
B bietet die Schnittstelle von PhoneTable an.

Dabei scheint aber die Implementierung als Tabelle durch.

Wenn A sie verwendet, kommt vielleicht der linke Teil heraus. Die Methoden im Interface müssen aufgerufen werden, was sonst.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Wenn es immer mehr Kunden werden, wird die Tabelle zu klein.

Eine andere Datenstruktur ist besser geeignet: die verkettete Liste.

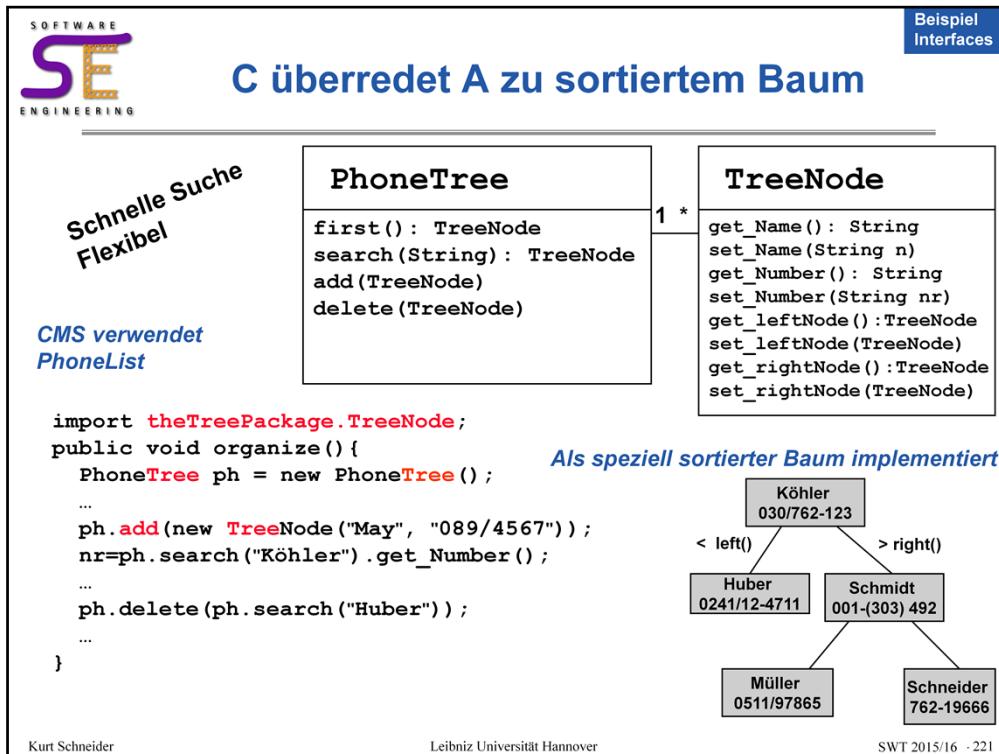
Auch hier scheint aber die Schnittstelle durch; daher muss A (links) sogar in diesen wenigen Zeilen schon einiges ändern, damit die gleiche Aufgabe nun mit der PhoneList statt der PhoneTable gelöst werden soll.

Ein echtes CMS wäre natürlich viel größer und müsste an hunderten Stellen Änderungen vornehmen – obwohl keine neuen Funktionen hinzugekommen sind.

Das ist ärgerlich und für A kaum verständlich.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Und jetzt passiert das Selbe noch einmal: Um effizienter (d.h. schneller beim Suchen) zu werden, verwendet B einen Baum statt der Liste.

Wieder ändert sich die Schnittstelle, wieder muss A überall die Methodenaufrufe ändern, wieder kostet es Zeit und Geld.

Das kann nicht die Lösung sein.

Dieses Phänomen gab und gibt es sehr, sehr häufig, und es ist sehr teuer.

Mit Information Hiding und konsequenter Verwendung von Schnittstellen kann man es vermeiden.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The slide features the SE Software Engineering logo in the top left. In the top right corner, there is a blue box labeled 'Beispiel Interfaces'. The main title 'Information Hiding: Interfaces!' is centered at the top. Below the title, a code block shows the Java interface `IPhoneBook` with methods `add`, `get_Number`, and `delete`. To the right of this is a box labeled 'Java Interface'. Below the interface definition, a code snippet shows its implementation in a class. A yellow callout bubble points from the implementation code to a note: 'oder: PhoneList()' and 'oder: PhoneTable(7)'. The note concludes with 'Das ist der einzige Unterschied!' (This is the only difference!). At the bottom of the slide, the author's name 'Kurt Schneider' and the university 'Leibniz Universität Hannover' are listed, along with the date 'SWT 2015/16 · 222'.

```
public void organize() {
    IPhoneBook ph;
    ph = new PhoneTree();
    ...
    ph.add("May", "089/4567");
    nr=ph.get_Number("Köhler");
    ...
    ph.delete("Huber");
    ...
}
```

oder: PhoneList()  
oder: PhoneTable(7)  
Das ist der einzige Unterschied!

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 · 222

iPhone hat nichts mit Apple zu tun, sondern mit der Konvention, ein Interface mit großem I vorne zu schreiben.

Diese Konvention ist inzwischen strittig, wird aber immer noch häufig verwendet (hier auch). Besser wäre es, die Nicht-Interface-Klassen mit `Impl` (für Implementation) zu beginnen.

Wie immer bei Konventionen: Legen Sie (in Ihrer Firma) sich auf eine fest. Welche, ist nicht so wichtig.

Aber das Wichtige hier ist:

Jetzt kann man alle drei gezeigten Implementierungen (Tabelle, Liste, Baum) verwenden, wenn man sich stets auf das gemeinsame Interface bezieht.

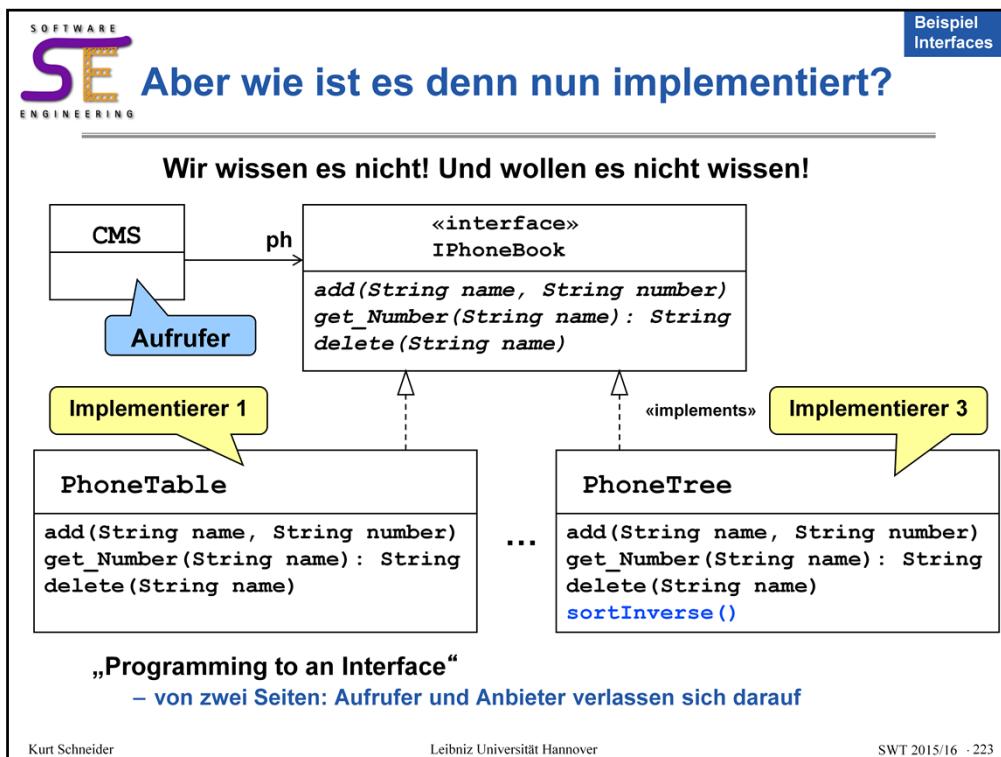
Das wird angegeben, um `ph` zu deklarieren. Dann kann das Objekt `ph` nur auf Methoden der Interfaceklasse `IPhoneBook` reagieren.

Darunter wird `ph` aber ein Objekt einer Unterklasse davon (also einer konkreten Realisierung) mit `new` zugewiesen.

Egal, welche Realisierung, die Interface-Methoden müssen alle Realisierungen kennen.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Phone Table und PhoneTree haben (blau) auch noch zusätzliche Methoden.

Die kann ph aber nicht ansprechen, weil sie nicht im Interface liegen. Sie sind also vor ph und anderen Verwendern „versteckt“, und das ist gut!

Damit wird man nirgends Methodenaufrufe ändern müssen, wenn man die drei (oder andere) Realisierungen gegeneinander austauscht.

Das ist erfreulich für CMS (bzw. A), weil alles gleich zu bleiben scheint, sogar, wenn B an den Interna herumoptimiert.

Und es ist erfreulich für B, weil er einfach neue Varianten liefern kann – A wird das nicht ärgern. B kann also in Ruhe weiter verbessern.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



## Was zeigt das Beispiel

- Abhängigkeiten zwischen Programmteilen entstehen leicht
  - Ändert sich eines, muss man das andere ändern
    - Aufwändig!
    - Fehleranfällig!
  - Was fehlte, war ein klarer Bezugspunkt: Schnittstelle/Interface
    - Zwischen Beteiligten ausgehandelt: „Contract“
- Interfaces
  - Sind syntaktische Einheit in Java
    - Abstrakte Klasse mit Methodenschnittstellen, ohne Rümpfe
    - Im Aufrufer dennoch als Typ verwenden `IPhoneBook`
    - Letztlich braucht man natürlich auch eine Implementierung `PhoneTree`, `PhoneList` oder `PhoneTable`
  - Vorteile
    - Nur Implementor-Name ist spezifisch im Aufrufer-Code
    - Aufrufer braucht bei Änderungen im Implementor nichts zu tun

Kurt Schneider                    Leibniz Universität Hannover                    SWT 2015/16 · 224

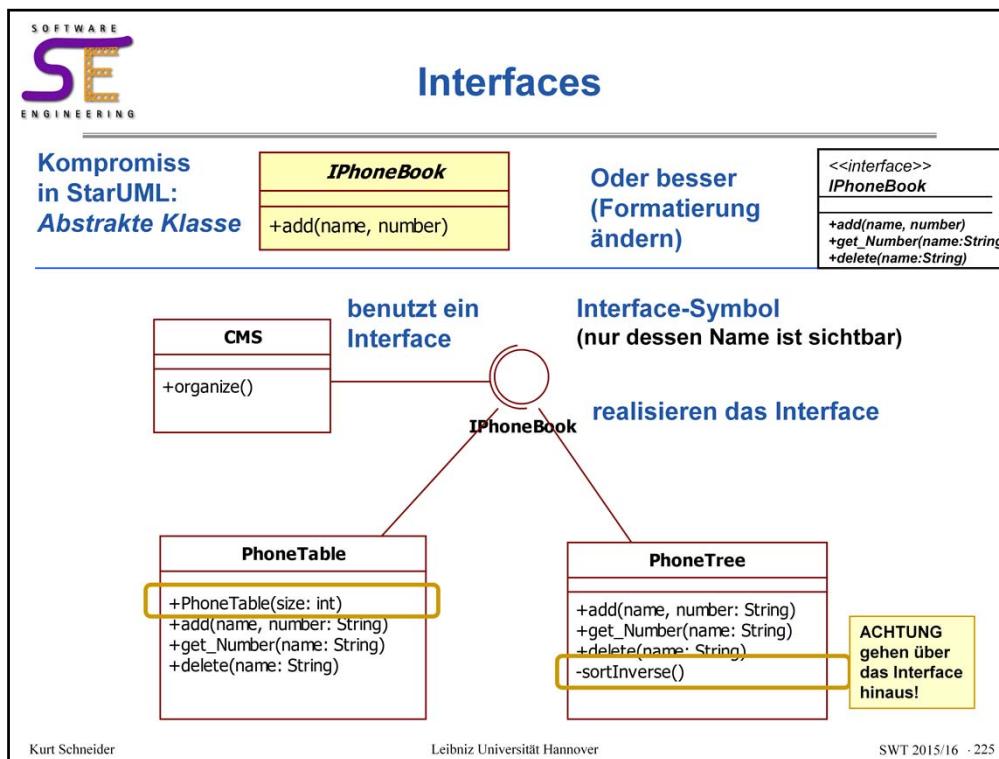
Der Schlüssel war die gemeinsame Verwendung eines Interface (Java-Schlüsselwort und –Begriff) als gemeinsamen Bezugspunkt.

Alle Details sind für die Aufrufer versteckt, und die Aufgerufenen müssen nur sicherstellen, dass sie die Interface-Methoden anbieten.

Weil das Interface als Oberklasse geerbt wird, stellt das schon Java sicher, sonst gibt es einen Compilerfehler.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Für Interfaces gibt es leider viele Varianten in den UML-Klassendiagrammen.

Die simpelste Variante ist der Kreis (angebotenes Interface) und die „greifende Hand“ (nachfragende Klasse, hier CMS).

In dieser Darstellung sieht man nur, wie das Interface heißt – aber schon die Methoden sieht man nicht, geschweige denn weitere Details.

Möchte man diese sehen, muss man das Interface als Klassensymbol (oben) darstellen.

UML erlaubt, darüber `interface` zu schreiben. Nicht jedes Tool kann das – StarUML zum Beispiel nicht.

Das ist ärgerlich, denn wir wollen uns ja nicht von einem Tool die Notation oder die Methode verbiegen lassen.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

```
SOFTWARE  
SE  
ENGINEERING  


## Interface in Java definieren



```
package de.unihannover.se.swt.phonedir;  
  
/**  
 * @author Kurt Schneider  
 */  
public interface IPhoneBook {  
    /**  
     * Add a name with its associate number to the phone book  
     *  
     * @param name      Name of a person  
     * @param number    Telephone number  
     */  
    public void add(String name, String number);  
  
    /**  
     * Retrieve the phone number associated with name  
     *  
     * @param name  
     * @return the number as a String, which may contain special characters  
     */  
    public String getNumber(String name);  
  
}  
Kurt Schneider
```


```

So geht es praktisch in Java (obwohl das Konzept der Schnittstellen völlig unabhängig von einer Programmiersprache ist! Das geht in vielen anderen Sprachen analog).

Hier sieht man, wie in einem Interface nur die Methodenköpfe definiert werden. Rümpfe gibt es nicht. Die müssen die Realisierungen nachliefern, jeweils anders.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover

The slide features the Software Engineering logo at the top left. The main title is "Implementierung zum Interface anbieten". Below the title is a code snippet for a class named PhoneList.

```
public class PhoneList implements IPhoneBook {  
    Node theList = null;  
    Node position = null; Als Liste implementieren  
  
    // appends a new entry (name, number) to the list  
  
    public void add(String name, String number) {  
        Node newNode = new Node();  
        newNode.setName(name);  
        newNode.setNumber(number);  
  
        this.append(newNode) Spezialmethoden  
wie append() oder  
sortInverse()  
nur intern verwenden,  
private machen.  
    }  
  
    usw.
```

Annotations in the code highlight specific parts:

- A callout box points to the line `theList = null;` with the text "Als Liste implementieren".
- A callout box points to the line `this.append(newNode)` with the text "Spezialmethoden wie append() oder sortInverse() nur intern verwenden, private machen.". This annotation also covers the entire block of code from the `add` method's definition to its closing brace.

At the bottom of the slide, there are three small text elements: "Kurt Schneider", "Leibniz Universität Hannover", and "SWT 2015/16 · 227".

So sieht eine Realisierung einer Methode in PhoneList aus.

Jede Realisierung erbt (extends) vom Interface, alle geerbten Methoden müssen implementiert werden – hier eben mit einer Liste.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



## Angebotene Methoden aufrufen

Nutzerprogramm, wie CMS

```
public void organize() {
    IPhoneBook ph;
    ph = new PhoneList()
    ...
    ph.add("May", "089/4567");
    nr=ph.getNumber("Köhler");
    ...
    ph.delete("Huber");
    ...
}
```

Hat Methoden wie IPhoneBook  
Implementiert als PhoneList

Kurt Schneider      Leibniz Universität Hannover      SWT 2015/16 · 228

Dann sieht man nur an einer Stelle (bei new), welche Realisierung gerade gewählt ist. Der gesamte Rest ist davon unabhängig.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



## Information Hiding Zusammenfassung

- Trennung von Schnittstelle und Implementierung
  - Schnittstelle muss ich kennen
  - Implementierung lieber nicht (information hiding)
- In Java: Spezielle Klassenkonstruktion
  - Ein Interface kann keine Instanzen haben
    - Kann ja nichts „tun“, spezifiziert nur
    - Etwas allgemeineres Konzept: „Abstrakte Klasse“
  - „Implementator“ erbt alle Methoden der Schnittstelle
    - Überschreibt sie, d.h.: implementiert, was da passiert
    - Kann zusätzliche Methoden haben; die nutzt Aufrufer lieber nicht
    - Nutzer bezieht sich nur auf das Interface
  - Vorteil von „Programming to an Interface“
    - Beide Seiten können sich unabhängig ändern
    - Probleme werden zerlegt

Kurt Schneider    Leibniz Universität Hannover                                      SWT 2015/16 · 229

Die Schnittstelle in Java dient dazu, im Interface zu definieren, wie „die Signatur“ aussehen soll. Unterklassen erben diese Schnittstellen, müssen sie aber noch konkret implementieren.

Da beide hier gezeigten grauen Implementierungen von derselben Schnittstelle geerbt haben, können sie auch gegeneinander ausgetauscht werden.

# Vorlesung „Grundlagen der Softwaretechnik (SWT)“ im WS 2015/2016

Prof. Dr. Kurt Schneider, FG Software Engineering, Leibniz Universität Hannover



Sieht gleich aus, aber wichtige Entwurfsentscheidungen sind verborgen: beim Smart kann man mit diesem Schaltknüppel auch Automatikgetriebe bedienen.