Vorlesung

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Programmieren I und II

Unit 6

Einführung in die objektorientierte Programmierung und Unified Modelling Language (UML)

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
1

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Prof. Dr. rer. nat.
Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

- **Raum: 17-0.10**
- **Tel.:** 0451 300 5549
- **Email: kratzke@fh-luebeck.de**

@NaneKratzke
Updates der Handouts auch über Twitter #prog_inf und #prog_itd

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
2

Units

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Unit 1 Einleitung und Grundbegriffe	**Unit 2** Grundelemente imperativer Programme	**Unit 3** Selbstdefinierbare Datentypen und Collections	**Unit 4** Einfache I/O Programmierung
Unit 5 Rekursive Programmierung und rekursive Datenstrukturen	**Unit 6** Einführung in die objektorientierte Programmierung und UML	**Unit 7** Konzepte objektorientierter Programmiersprachen	**Unit 8** Testen (objektorientierter) Programme
Unit 9 Generische Datentypen	**Unit 10** Objektorientierter Entwurf und objektorientierte Designprinzipien	**Unit 11** Graphical User Interfaces	**Unit 12** Multithread Programmierung
Unit 13 Einführung in die Funktionale Programmierung			

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
3

Abgedeckte Ziele dieser UNIT

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Kennen existierender Programmierparadigmen und Laufzeitmodelle	Sicheres Anwenden grundlegender programmiersprachlicher Konzepte (Datentypen, Variable, Operatoren, Ausdrücke, Kontrollstrukturen)	Fähigkeit zur problemorientierten Definition und Nutzung von Routinen und Referenztypen (insbesondere Liste, Stack, Mapping)	Verstehen des Unterschieds zwischen Werte- und Referenzsemantik
Kennen und Anwenden des Prinzips der rekursiven Programmierung und rekursiver Datenstrukturen	Kennen des Algorithmus-begriffs, Implementieren einfacher Algorithmen	Kennen objektorientierter Konzepte Datenkapselung, Polymorphie und Vererbung	Sicheres Anwenden programmiersprachlicher Konzepte der Objektorientierung (Klassen und Objekte, Schnittstellen und Generics, Streams, GUI und MVC)
Kennen von UML Klassendiagrammen, sicheres Übersetzen von UML Klassendiagrammen in Java (und von Java in UML)	Kennen der Grenzen des Testens von Software und erste Erfahrungen im Testen (objektorientierter) Software	Sammeln erster Erfahrungen in der Anwendung objektorientierter Entwurfsprinzipien	Sammeln von Erfahrungen mit weiteren Programmiermodellen und -paradigmen, insbesondere Mulithread Programmierung sowie funktionale Programmierung

Am Beispiel der Sprache JAVA

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
4

Themen dieser Unit

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Warum eigentlich OO?

- Beherrschung von Komplexität
- Kapselung
- Polymorphie
- Vererbung

Objektorientierte Konzepte

- Denken in Objekten
- Information Hiding
- Hierarchiebildung
  - Vererbung
  - Zerlegung
- Objektkommunikation und Assoziationen

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
5

Zum Nachlesen ...

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Kapitel 1

Einleitung

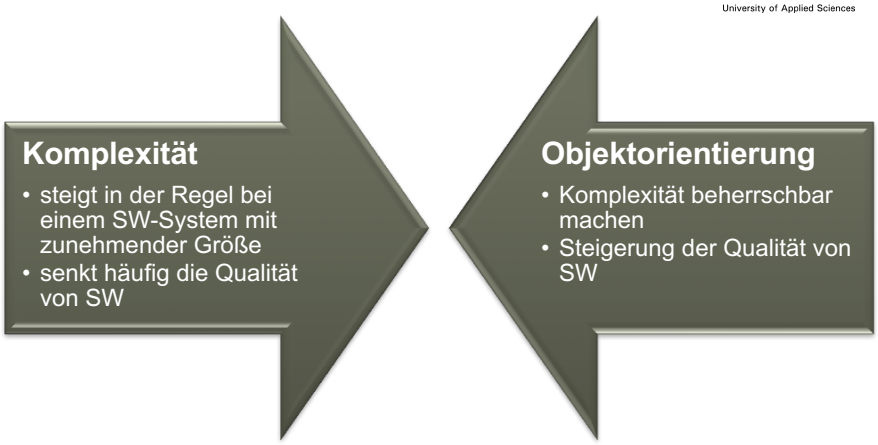
Kapitel 2

Die Basis der Objektorientierung

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
6

Objektorientierung als Mittel zur Beherrschung von Komplexität

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



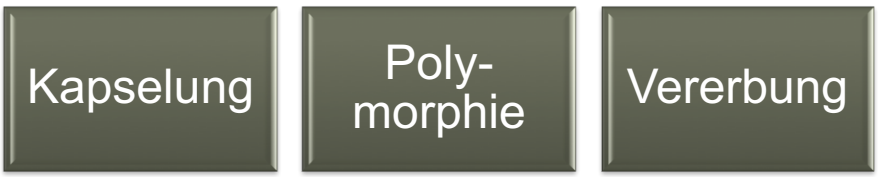
„Die Techniken der objektorientierten SW-Entwicklung unterstützen [...] dabei, Software einfacher erweiterbar, besser testbar und besser wartbar zu machen."

[LR09, S. 27]

Grundelemente der Objektorientierung

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- **Objektorientierung** kann als ein Werkzeugkasten verstanden werden, um die Zielsetzungen der Entwicklung von Software anzugehen.
- **Basiswerkzeuge** sind:



Vorläufer der objektorientierten Programmierung

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- **Prozedurale Programmierung**
- Ausgangspunkt Inhalt eines Computerspeichers
  - Daten
  - Instruktionen

Strukturierung von Instruktionen	Strukturierung von Daten
• Verzweigungen • Zyklen • Routinen mit Aufruf- und Rückgabeparametern	• Datentypen • Zeiger, Records, Arrays, Listen, Bäume, Mengen

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
9

Prozedurale Programmierung

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Typische (prozedurale) Programmiersprachen	Objektorientierte Erweiterungen

• C
• Pascal
• Fortran
• COBOL
	• Kapselung von Daten
	• Polymorphie
	• Vererbung
	• Bspw: geboten durch
	  • C++, C#
	  • JAVA
	  • Python
	  • PHP

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
10

Verantwortlichkeit des Entwicklers bei prozeduralen Programmiersprachen

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Das dies manchmal nicht funktioniert, lassen manche C Programme vermuten.

- ProgrammiererIn hat volle Kontrolle welche Routinen, welche Daten aufrufen.

Kontrolle

- ProgrammiererIn hat auch die Verantwortung, dass die richtigen Routinen die richtigen Daten nutzen.

Verantwortung

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
11

Grundelemente der Objektorientierung

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- **Objektorientierung** kann als ein Werkzeugkasten verstanden werden, um die Zielsetzungen der Entwicklung von Software anzugehen.
- **Basiswerkzeuge** sind:

Kapselung | Poly-morphie | Vererbung

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
12

Kapselung von Daten

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Daten gehören einem Objekt

↓

Kein direkter Zugriff auf Daten

↓

Datenzugriff grundsätzlich nur über Methoden eines Objekts

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
13

Hintergrund der Datenkapselung

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Objekt sorgt für Konsistenz seiner Daten
- dient dem Zwecke:

- Konsistenz der Daten einfacher sicherzustellen
- Reduktion des Aufwands von Änderungen
- Änderungen lassen sich auf Einzelobjekte (bzw. deren Klassen) beschränken

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
14

Prinzip der Kapselung

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Daten	Methoden
• Satz von Variablen • Für jedes Objekt neu angelegt (**Instanzvariablen**) • Instanzvariablen repräsentieren den **Zustand** eines Objekts • Zustand eines Objekts kann sich während Lebensdauer ändern • Zugriff kann eingeschränkt werden	• Auf Daten operierende Routinen • **Methoden** nur einmal vorhanden • Methoden **operieren** aber **auf Instanzvariablen** • Methoden definieren das Verhalten eines Objekts • Zugriff auf Methoden kann eingeschränkt werden

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
15

Daten- und Methodensichtbarkeiten
`public, protected und private`

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Details folgen ...

```
class An_Object {

  public Object forall;

  protected Object forchildren;

  private Object my_eyes_only;

  public Object public_method() {};

  protected Object protected_method() {};

  private Object private_method() {};

}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
16

Daten- und Methodensichtbarkeiten
`public, protected und private`

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Daten- und Methodensichtbarkeiten können dazu genutzt werden

- Daten zu verbergen (zu kapseln)
- Datenzugriffe einzuschränken
- Datenzugriffe nur über definierte Schnittstellen zuzulassen.

- Code zu verbergen (zu kapseln)
- Codeaufrufe einzuschränken
- Codebereiche festzulegen, die für zukünftige Anpassungen gesperrt sind.
- Codebereiche festzulegen, in denen zukünftige Anpassungen stattzufinden haben.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
17

Grundelemente der Objektorientierung

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- **Objektorientierung** kann als ein Werkzeugkasten verstanden werden, um die Zielsetzungen der Entwicklung von Software anzugehen.
- **Basiswerkzeuge** sind:

Kapselung | Poly-morphie | Vererbung

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
18

Prinzip der Polymorphie

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Polymorphie bedeutet im Wortsinne „Vielgestaltigkeit“

Bsp.: Fassung und Leuchtmittel

Standardisierte Fassungen arbeiten sowohl mit

Klassischen Glühbirnen	Energie-sparlampen	LED-Lampen

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
19

Prinzip der Polymorphie

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Einheitliche Schnittstellen
- unterschiedliche Ausprägungen von Funktionalitäten
- dient dem Zwecke:

- Bereiche im Code für „Plugins“
- Wiederverwendbarkeit von „Meta“funktionalitäten
- Wesentlich flexiblere Software
- Steigerung der Wartbarkeit und Änderbarkeit

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
20

Beispiel für Polymorphie
Sortieren (hier: klassischer bubbleSort)

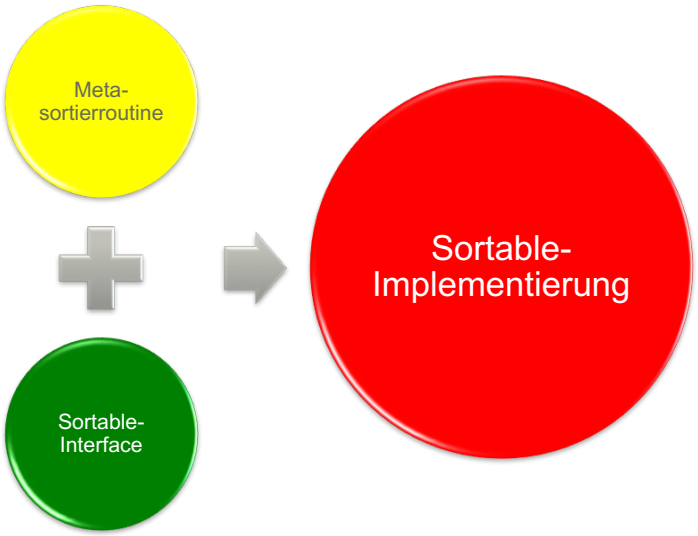
FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
public static void bubbleSort(int[] xs) {
  boolean unsorted=true;

  while (unsorted) {
    unsorted = false;
    for (int i=0; i < xs.length-1; i++) {
      if (!xs[i] <= xs[i+1]) {
        swap(xs[i], xs[i+1]);
        unsorted = true;
      }
    }
  }
}
```

Was fällt auf ?

Polymorpher BubbleSort

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



Polymorpher BubbleSort
„Meta“-Sortierroutine bubbleSort

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
public static void bubbleSort(ISortable[] xs)
{
  boolean unsorted=true;

  while (unsorted) {
    unsorted = false;
    for (int i=0; i < xs.length-1; i++) {
      if (!xs[i].is_sorted(xs[i+1])) {
        swap(xs[i], xs[i+1]);
        unsorted = true;
      }
    }
  }
}
```

Meta-sortierroutine

Polymorpher BubbleSort
Sortable Schnittstelle für bubbleSort

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
public interface ISortable {
  public boolean is_sorted(ISortable compare);
  public Object value();
}
```

Sortable-Interface

Polymorpher BubbleSort
Mittels BubbleSort sortierbares Objekt

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
public class Car implements ISortable {

  private int horsepower;

  public Car(int horsepower) {
    this.horsepower = horsepower;
  }

  public boolean is_sorted(ISortable compare){
    return (Integer)this.value() < (Integer)
      compare.value() ? true : false;
  }

  public Object value() {
    return this.horsepower;
  }
}
```

Imple-mentierung

Polymorpher BubbleSort

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

So gestaltete Sortieralgorithmen ermöglichen es, verschiedenste Objekte nach definierbaren Kriterien z[illegible]tieren, z.B.

Details zu Polymorphie folgen in Unit 6 und 7

Autos nach	Flugzeuge nach	Akti[illegible] nach	[illegible] nach
Pferdestärken	Geschwin-digkeit	KGV	...
Sprit-verbrauch	Dienst-gipfelhöhe	Marktkapitali-sierung	...
...	...	...	...

Grundelemente der Objektorientierung

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- **Objektorientierung** kann als ein Werkzeugkasten verstanden werden, um die Zielsetzungen der Entwicklung von Software anzugehen.
- **Basiswerkzeuge** sind:

Kapselung	Poly-morphie	Vererbung

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
27

Zwei Formen der Vererbung
Beispiele aus dem Alltag

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Erben einer Spezifikation (Standardisierung)	Erben von Umsetzungen (Implementierung)
• Leuchtmittel können ganz unterschiedliche Formen und Technologien haben. • Sie können dennoch in derselben Fassung betrieben werden. • Grundlegende Gemeinsamkeit: Spezifikation (genormte Fassung, 220 V Wechselstrom).	• Die EU legt rechtliche Rahmenbedingungen fest • Deutschland hat gesetzliche Regelungen für das Steuerrecht. • Das Land Schleswig-Holstein hat spezielle Regelungen. • Lübeck legt eigene Regelungen fest, bspw. Hebesatz für Gewerbesteuer.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
28

Beispiel: Erben einer Umsetzung

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
class Auto {
  protected double KMStand;
  protected boolean unfall;

  double getKMStand() {
    return this.KMStand;
  }

  boolean unfallschaden() {
    return this.unfall;
  }
}
```

Ursprungs-klasse

Überschreiben von Methoden unter Nutzung der ursprünglichen Semantik

Überschreiben von Methoden

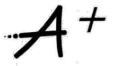
Erweitern um Methoden, Daten

Abgeleitete Klasse

```
class BilligheimerAuto extends Auto {

  double getKMStand() {
    return 0.75 * super.getKMStand();
  }

  boolean unfallschaden() {
    return FALSE;
  }

  String angebotsText() {
    return „Einmaliges
      Supersonderangebot!“;
  }
}
```

Details folgen ...

Zusammenfassung



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Objektorientierung ist ein Art Werkzeugkasten, um die Entwicklung und Wiederverwendung von Software zu optimieren (steigende Komplexität größerer SW-Systeme zu beherrschen)
- Einleitung in die Kernkonzepte der Objektorientierung

- **Einheit von**
  - **Daten (Zustand eines Objekts) und**
  - **Code (Verhalten eines Objekts)**
- **Kapselung**
- **Polymorphie**
- **Vererbung**

Themen dieser Unit

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Warum eigentlich OO?

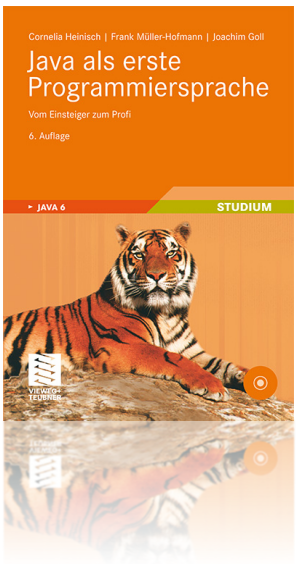
- Beherrschung von Komplexität
- Kapselung
- Polymorphie
- Vererbung

Objektorientierte Konzepte

- Denken in Objekten
- Information Hiding
- Hierarchiebildung
  - Vererbung
  - Zerlegung
- Objektkommunikation und Assoziationen

Zum Nachlesen ...

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



Kapitel 2

Objektorientierte Konzepte

2.1 Modellierung mit Klassen und Objekten

2.2 Das Konzept der Kapselung

2.3 Abstraktion und Brechung der Komplexität

Noch mehr zum Nachlesen ...

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Kapitel 4

UML Grundlagen

4.3.1 Klasse

4.4.1 Generalisierung, Spezialisierung

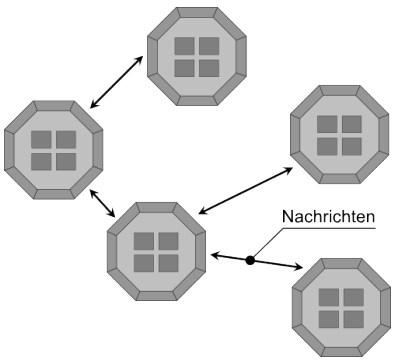
4.4.2 – 4.4.5 Assoziation (gerichtet, attributiert, qualifiziert)

4.4.7 – 4.4.8 Aggregation und Komposition

Modellierung mit Klassen und Objekten

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Entscheidend für den objektorientierten Ansatz, ist nicht das objektorientierte Programmieren,
- sondern das Denken in Objekten
- Bei der objektorientierten Modellierung denkt man lange Zeit hauptsächlich im Problembereich



Modellierung mit Klassen und Objekten

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Modellierung des Problembereichs mittels
  - Problembereichs-spezifischer Datentypen
  - in Form von Klassen
- Eine Klasse ist ein Gegenstand der realen Welt (Entität) und kann sein

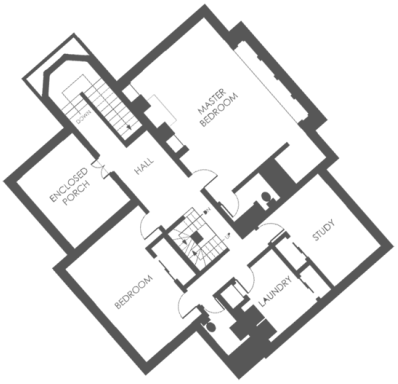
Entität		
Gegenstand	Wesen	Konzept
• z.B. Auto • Stuhl	• z.B. Mitarbeiter	• z.B. Vertrag, • Lehrveranstaltung

- Der Ansatz der Objektorientierung beruht darauf, Objekte der realen Welt mittels softwaretechnischer Mittel abzubilden.

Klassen und Objekte (I)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Klassen stellen die **Baupläne** für Objekte dar.
- **Klassen** sind die **Datentypen**
- **Objekte** die **Instanzen** dieser Datentypen.
- Objekte werden gemäß den in den Klassen abgelegten Bauplänen erzeugt.



Klassen und Objekte (II)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Eine Klasse
  - trägt einen **Klassennamen**
  - enthält **Datenfelder** (Attribute)
  - und **Methoden**, die auf diese Klasse zugreifen.

Punkt	
x : int y : int	Klassenname `Punkt` Datenfeld `x` vom Typ `int` Datenfeld `y` vom Typ `int`
zeichne( ) verschiebe( ) loesche( )	Methode `zeichne()` Methode `verschiebe()` Methode `loesche()`

Darstellung einer Klasse mittels UML

Exkurs: UML
Unified Modelling Language

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Die Unified Modeling Language (UML) ist eine graphische Modellierungssprache zur
  - Spezifikation,
  - Konstruktion und
  - Dokumentation von (objektorientierter) Software
- UML hat sich insbesondere im OO-Umfeld als Quasistandard etabliert
- UML definiert graphische Notationen (Diagramme) für statische Strukturen und dynamischen Abläufen
- UML wird von der Object Management Group (OMG) entwickelt und ist zertifzierter ISO Standard (ISO/IEC 19501)

OBJECT MANAGEMENT GROUP

Exkurs UML: Diagrammarten

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Strukturdiagramme	Verhaltensdiagramme
• **Klassendiagramme**	• **Aktivitätsdiagramm**
• Montagediagramm	• Use Case Diagramm
• Komponentendiagramm	• Interaktionsübersichtsdiagramm
• Verteilungsdiagramm	• Kommunikationsdiagramm
• Objektdiagramm	• **Sequenzdiagramm**
• Profildiagramm	• Zeitverlaufsdiagramm
	• **Zustandsdiagramm**

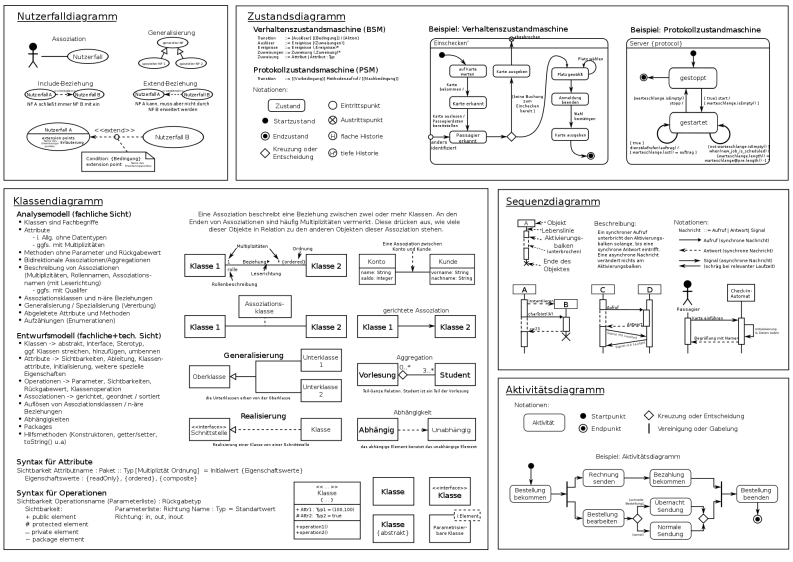
UML kennt die oben stehenden Diagrammarten. Die markierten Diagramme sind die gebräuchlichsten und werden im Rahmen der Vorlesung genutzt.

Die grafische UML-Notation wird Stück für Stück an den geeigneten Stellen im Verlaufe der Vorlesung eingeführt.

Exkurs UML: Diagramm Übersicht nur zur Information

FACH HOCHSCHULE LÜBECK

http://de.wikipedia.org/wiki/Unified_Modeling_Language



Klassen und Objekte (III)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Punkt
x : int y : int
zeichne( ) verschiebe( ) loesche( )

Klassenname `Punkt`
Datenfeld `x` vom Typ `int`
Datenfeld `y` vom Typ `int`
Methode `zeichne()`
Methode `verschiebe()`
Methode `loesche()`

```
class Punkt {

  int x;
  int y;

  void zeichne() { ... }
  void verschiebe() { ... }
  void loesche() { ... }

}
```

UML **JAVA**

Selber Sachverhalt – andere Notation

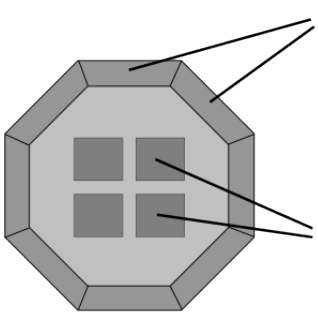
Im Rahmen dieser Vorlesung wird UML primär zur Darstellung struktureller oder ablauf-orientierter Sachverhalte genutzt und JAVA für programmiertechnische Implementierungen.

Beide Formen werden aber parallel genutzt. Lauffähig programmieren lässt sich übrigens nur in JAVA.

Klassen und Objekte (IV)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Bei der Objektorientierung werden die
  - **Daten** eines Objektes und
  - Die Daten verändernden **Methoden**
  - als eine **Einheit** betrachtet – das **Objekt**.



Klassen und Objekte (V)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- **Methoden** erfüllen die Aufgaben
  - Werte der Datenfelder **auszugeben**
  - Datenfelder zu **verändern**
  - Mittels in Datenfeldern gespeicherte Werte neue Ergebnisse zu **berechnen**
- **Datenfelder** definieren mögliche **Zustände** der Objekte (Datenstruktur),
- die **Methoden** bestimmen das **Verhalten** der Objekte.

Zustände von Objekten

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Ein Objekt hat einen Satz von Datenfeldern (und Methoden)
- Jedes Datenfeld hat Werte
- **Zustand eines Objekts == momentane Wertebelegung der Datenfelder des Objekts**
- **Beispiel Fahrstuhl**
  - Gewichtssensor im Fahrstuhl
  - **Mikroskopischer Zustand** des Fahrstuhls == aktueller Wert des Sensors
  - **Makroskopischer Zustand** des Fahrstuhls == Überladen oder nicht Überladen

Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Gegeben ist folgende Klassendefinition.

```
class Auto {
  private double fuel = 0.0;
  private double kmstand = 0.0;

  public Auto() {
    this.fuel = 5.0;
  }

  public tanke(double l) {
    this.fuel += l;
  }

  public void fahre(double km) {
    this.kmstand += km;
    this.fuel -= 7.0 * km / 100;
  }
}
```

Geben Sie den Mikrozustand des erzeugten Objekts nach den entsprechenden Methodenaufrufen an.

```
Auto car = new Auto();
```

```
car.tanke(50.0);
```

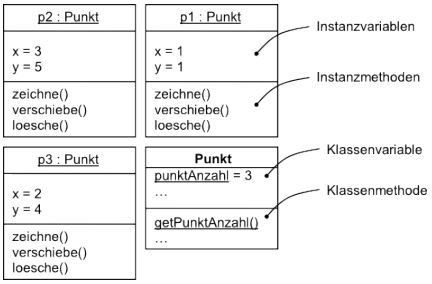
```
car.fahre(50.0);
```

```
car.fahre(200.0);
car.tanke(10.0);
```


Instanzen- und Klassenvariablen/-methoden

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Drei Objekte p1, p2, p3 der Klasse Punkt
- Jeder Punkt hat individuelle Koordinaten (**Instanzvariablen**)
- Methoden **zeichne(), verschiebe()** und **loesche()** arbeiten auf Instanzvariablen und heißen daher **Instanzmethoden**.



- Anzahl der erzeugten Punkte ist jedoch eine Eigenschaft der Menge aller Punkte
- Die Variable **punktAnzahl** wird **Klassenvariable** genannt und wird in der Klasse einmalig gespeichert
- Methoden die auf Klassenvariablen zugreifen werden **Klassenmethoden** genannt

*Klassenvariablen und -methoden werden in **UML** unterstrichen*

*Objekte werden in **UML** durch ein : vom Klassennamen getrennt o : T (zu lesen als o ist vom Typ T) und unterstrichen*

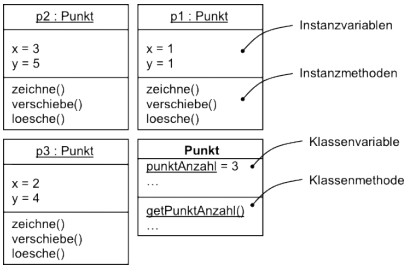
Instanzen- und Klassenvariablen

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Variable, die allen Instanzen einer Klasse gemeinsam sind, werden in der Objektorientierung als Klassenvariable bezeichnet
- **Klassenvariable** stellen **globale Variable** für alle Objekte einer Klasse dar
- Klassenvariable werden in der Klasse selbst als **Unikat für alle Objekte** der Klasse angelegt
- In JAVA nutzt man hierzu das Schlüsselwort **static**
- **Variable** die bei jedem Objekt eine individuelle Ausprägung annehmen können, werden als Instanzvariablen bezeichnet

Instanzen- und Klassenvariablen
UML und JAVA

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



*Definition der Klasse Punkt und drei Objekte p1, p2, p3 in **UML***

```
class Punkt {
  int x;
  int y;
  static int punktAnzahl;

  void zeichne() { ... }
  void verschiebe() { ... }
  void loesche() { ... }

  static int getPunktAnzahl() { ... }
}
```

*Definition der Klasse Punkt und drei Objekte p1, p2, p3 in **JAVA***

```
p1 = new Punkt();
p2 = new Punkt();
p3 = new Punkt();
```

Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Sie sollen nun Personen wie folgt anlegen können.

```
Person p1 = new Person("Max", "Mustermann");
Person p2 = new Person("Maren", "Musterfrau");
Person p3 = new Person("Tessa", "Loniki");
```

Geben Sie nun die angelegten Personen wie folgt aus,

```
System.out.println(p2);
System.out.println(p1);
System.out.println(p3);
```

soll folgendes auf der Konsole ausgegeben werden.

```
Maren Musterfrau (2 von 3 angelegten Personen)
Max Mustermann (1 von 3 angelegten Personen)
Tessa Loniki (3 von 3 angelegten Personen)
```

Bitte geben Sie eine Implementierung für `Person` an, die „mitzählen“ kann, wie viele Personenobjekte bereits angelegt wurden.

Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Lösung:

Miniübung:

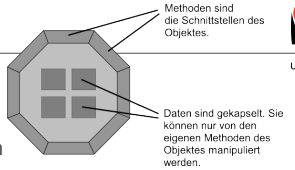


FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Geben Sie nun ein UML-Klassendiagramm an, dass ihre Lösung beschreibt:

```
public class Person {
  private String vorname;
  private String nachname;
  private int nr;
  private static int total;

  public Person(String vn, String nn) {
    this.vorname = vn; this.nachname = nn;
    this.nr = Person.total + 1;
    Person.total++;
  }

  public String toString() {
    return vorname + " " + nachname +
           " (" + nr + ". von " +
           Person.total + " Personen)";
  }
}
```


Konzept der Kapselung



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

In der Objektorientierung betrachtet man Daten und Methoden als eine zusammengehörende Einheit. Die folgenden Begriffe sind dabei von Bedeutung:

Abstraktion	Kapselung	Information Hiding
• Komplexer Sachverhalt der realen Welt • wird auf das Wesentliche reduziert • und vereinfacht dargestellt • Datenfelder und Methoden eines Objekts repräsentieren diejenigen Daten und das Verhalten von Bedeutung für den Problemraum	• Objekt implementiert sein Verhalten in Schnittstellenmethoden • Ein Objekt sollte (im Idealfall) nur über definierte Schnittstellenmethoden mit seiner Umwelt in Kontakt treten	• Innere Daten eines Objekts sollen nach außen nicht direkt sichtbar sein • Innere Eigenschaften eines Objekts sollen verborgen sein • Ein Objekt sollten nichts von inneren Implementierungs-details eines anderen Objekts wissen müssen

Ein Objekt sollte also keine Kenntnisse über den inneren Aufbau anderer Objekte haben. Programmiertechnische Änderungen innerhalb von Klassen (und daraus instantiierten Objekten) ziehen so keine Änderungen außerhalb der geänderten Klassen nach sich, solange die Schnittstellen gleich bleiben.

Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Sie sollen nun Personen weiterhin wie folgt anlegen können.

```
Person p1 = new Person("Max", "Mustermann");
Person p2 = new Person("Maren", "Musterfrau");
Person p3 = new Person("Tessa", "Loniki");
```

Jedoch auf die einzelnen Namensbestandteile zielgerichtet zugreifen können.

```
System.out.println(p2.getNachname());
System.out.println(p1.getVorname());
System.out.println(p3.getVorname() + " " + p3.getNachname());
```

Es soll folgendes auf der Konsole ausgegeben werden.

```
Musterfrau
Max
Tessa Loniki
```

Bitte geben Sie eine Implementierung für `Person` an, die entsprechende getter Methoden implementiert.

Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Sie sollen nun Personen weiterhin wie folgt anlegen können.

```
Person p1 = new Person("Max", "Mustermann");
Person p2 = new Person("Maren", "Musterfrau");
Person p3 = new Person("Tessa", "Loniki");
```

Jedoch im nachhinein Nachnamen sinnvoll ändern können.

```
p2.setNachname("Mustermann");
System.out.println(p2);
```

Es soll dann folgendes auf der Konsole ausgegeben werden.

```
Maren Mustermann
```

Werden sinnlose Werte wie "" oder null als Nachname gesetzt, soll nichts im Objekt geändert werden. Die Methode soll aber `false` als Rückgabe liefern. Wird etwas geändert, soll sie `true` liefern.

```
p1.setNachname("") == false      => p1 bleibt Max Mustermann
p1.setNachname(null) == false    => p1 bleibt Max Mustermann
p1.setNachname("Müller") == true => p1 wird Max Müller
```


Miniübung:



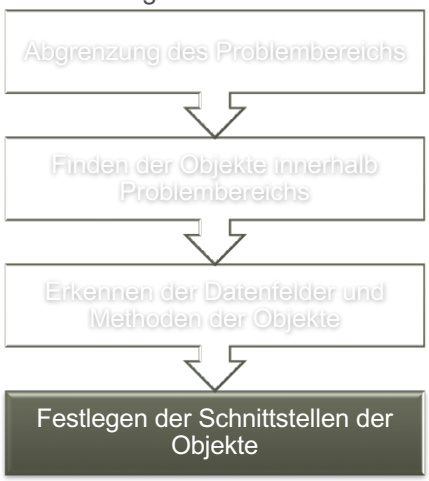
FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Abstraktion und Brechung der Komplexität

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

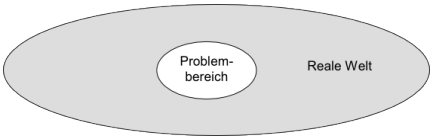
- Abstraktion
- Wesentliches erkennen
- Unwesentliches weglassen
- Mittel um Komplexität eines Systems beherrschbar zu machen

Übliches Vorgehen in der OO-Entwicklung



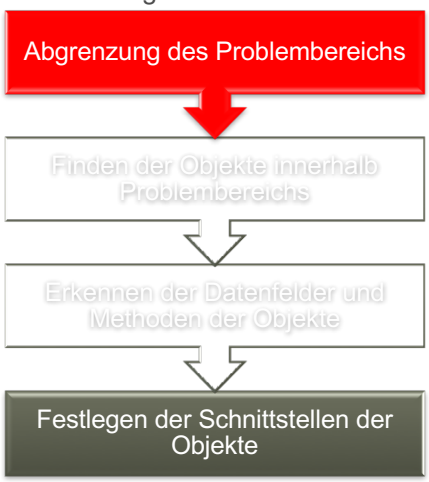
Abstraktion zur Abgrenzung des Problembereichs

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



- Ausschnitt der realen Welt
- Problembereich, der analysiert werden soll
- Bereich der zu untersuchenden Aufgaben
- Teil der realen Welt, der durch die Software abgedeckt/unterstützt werden soll

Übliches Vorgehen in der OO-Entwicklung

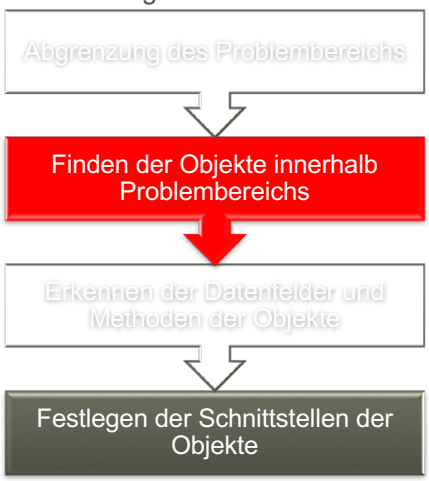


Abstraktion zum Finden der Objekte im Problembereich

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Relevante Objekte innerhalb des Problembereichs finden
- Entitäten der realen Welt auf Objekte der Programmiersprache abbilden
- Kernfrage: Ist es notwendig für eine Entität des Problembereichs Informationen im Programmsystem zu führen?
- Ja => Ableiten einer Klasse
- Nein => „Wegabstrahieren“

Übliches Vorgehen in der OO-Entwicklung



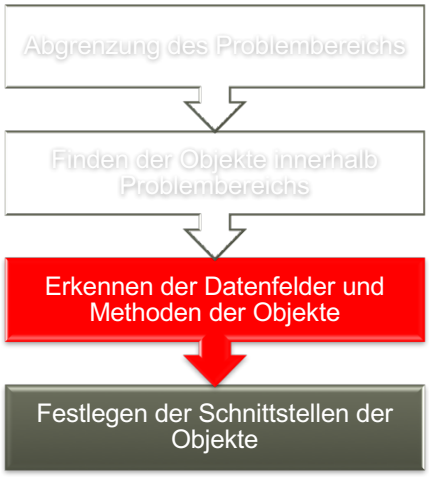
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
59

Abstraktion und Brechung der Komplexität (III)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Welche Informationen über ein Objekt sind erforderlich?
- Sehr stark abhängig von der Anwendung und den Anforderungen an die Software
- Autohersteller wird andere Daten über ein Auto benötigen, als ein Finanzamt

Übliches Vorgehen in der OO-Entwicklung



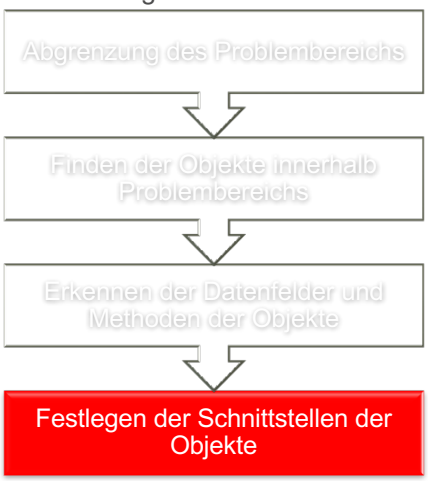
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
60

Abstraktion zur Festlegen von Schnittstellen der Objekte

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Welche Daten eines Objekts müssen nach außen sichtbar sein?
- Welche Methoden müssen nach außen sichtbar sein?
- Welche Daten, welches Verhalten muss besonders „geschützt“ werden?
- Aus der Beantwortung dieser Fragen wird die Ableitung einer Schnittstelle für ein Objekt (Klasse) vorgenommen.

Übliches Vorgehen in der OO-Entwicklung



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
61

Abstraktion zur Bildung von Hierarchien

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

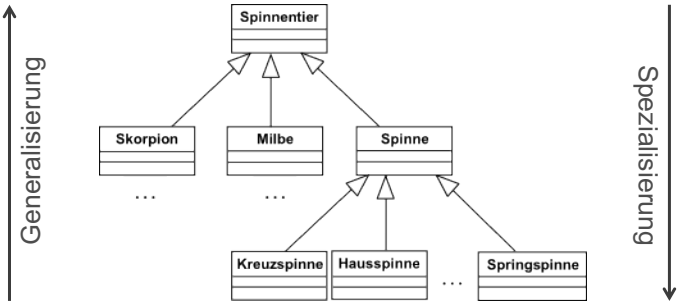
- Abstraktion und Information Hiding sind effiziente Mittel um Komplexität zu beherrschen
- Ein weiteres Mittel ist die Bildung von **Hierarchien**
- Die Objektorientierung kennt im Kern zwei Hierarchieformen:

Vererbungshierarchie	Zerlegungshierarchie
• Kind of-Hierarchie • Is a-Hierarchie • Anordnung von Klassen in Kategorieebenen(-bäumen)	• Part of-Hierarchie • Betrachtung von zusammengesetzten Objekten in Form von • Aggregationen • Kompositionen

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
62

Vererbungshierarchien (I)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



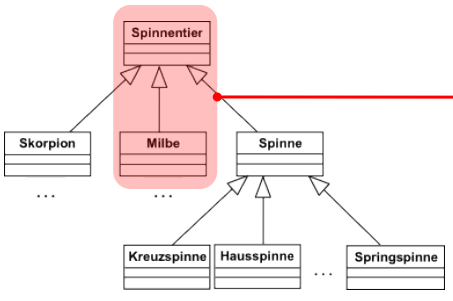
Darstellung von Vererbungshierarchien in UML:

Pfeil bedeutet bspw. Skorpion ist Unterklasse von Spinnentier

Kann auch so gelesen werden: Skorpion (spezieller) ist ein Spinnentier (genereller), daher auch der Name „is a-Hierarchie“

Vererbungshierarchien (II)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



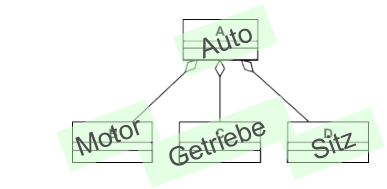
Darstellung von Vererbungshierarchien in UML

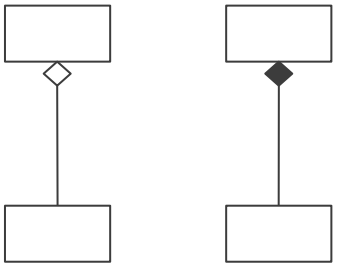
```
class Spinnentier {
  ...
}

class Milbe extends Spinnentier {
  ...
}
```

Ausdrücken einer Vererbung in JAVA (nur der markierte Ausschnitt)

Zerlegungshierarchie

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

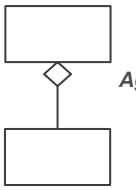




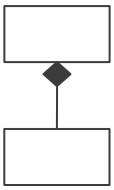
- Ein Objekt kann als Datenfelder andere Objekte haben
- Z.B. ein Auto besteht aus einem Motor, Getriebe und Sitzen (sowie weiteren Teilen)
- Man kann ein Objekt in seine Teilobjekte und diese wiederum in ihre Teilobjekte zerlegen (usw.).
- Bei dieser Zerlegung unterscheidet man Aggregationen und Kompositionen (kommt gleich)

Zerlegungshierarchie
Aggregation und Komposition (Spezialformen von Assoziationen)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



Aggregation in UML



Komposition in UML

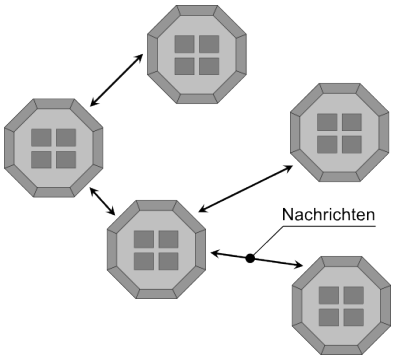
- Bei einer Aggregation können die Bestandteile eines Objekts unabhängig von der Lebensdauer des Oberobjekts existieren
- **Teile können länger leben als das Ganze**
- **Beispiel:** Die Räder eines Autos können an andere Autos gebaut werden. Räder sind an ein Auto aggregiert (zugeordnet).

- Bei einer Komposition existieren die Bestandteile eines Objekts nur so lange wie auch das Oberobjekt existiert.
- **Teile können nicht länger leben als das Ganze**
- **Beispiel:** Die Seiten eines Buchs sind mit dem Buch untrennbar verbunden. Seiten und Buch sind komponiert.

Zusammenarbeit von Objekten
Objektkommunikation

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

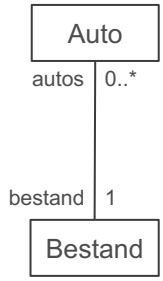
- Objektorientierte Systeme erbringen ihre Leistung durch das Zusammenwirken von Objekten
- in dem Nachrichten zwischen Objekten ausgetauscht werden
- (in JAVA entspricht dies Methodenaufrufen)



Assoziation zwischen Objekten

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Assoziation in JAVA




```
class Auto {
  Bestand bestand; // Verweist auf einen Bestand
  ...
}
```

```
class Bestand {
  List<Auto> autos = new LinkedList<Auto>();
  // Verweist auf eine Liste von Autos
  ...
}
```

Assoziationen sind erforderlich, damit Objekte miteinander Kommunizieren können (d.h. Kenntnis voneinander haben).

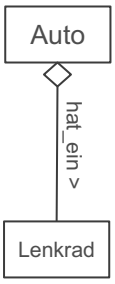
Programmiertechnisch, wird üblicherweise eine Assoziation mit Hilfe zweier Datenfelder erzeugt, die Referenzen zwischen den Objekten halten.

- Für die Konnektivitäten **0..1** (oder oder eine Verbindung) und **1** (genau eine Verbindung) kann dabei einfach eine einfache Referenzvariable genutzt werden.
- Für Konnektivitäten **> 1** muss eine Datenstruktur gewählt werden, die mehr als einen Verweis aufnehmen kann. Üblicherweise wird hier eine Liste/Array genutzt.

Aggregation/Komposition in UML/JAVA

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Aggregation in UML



Aggregation in JAVA

```
class Auto {
  Lenkrad hat_ein;
  ...
}
```

```
class Lenkrad {
  ...
}
```

```
Auto auto = new Auto();
Lenkrad lenkrad = new Lenkrad();
auto.hat_ein = lenkrad;
```

Programmiertechnisch, wird üblicherweise eine Aggregation/Komposition mit Hilfe einer Variablen erzeugt, die eine Referenz auf das Teilobjekt enthält. Da JAVA nur Referenztypen kennt, geht dies in JAVA sehr einfach (siehe oben). Solch eine Variable wird auch **Referenzvariable** (ergänzend zu Instanz- und Klassenvariable genannt).

Kompositionen werden in der Regel genauso umgesetzt, aber beim Löschen wird auch das Komposit mitgelöscht.

Multiplizitäten

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

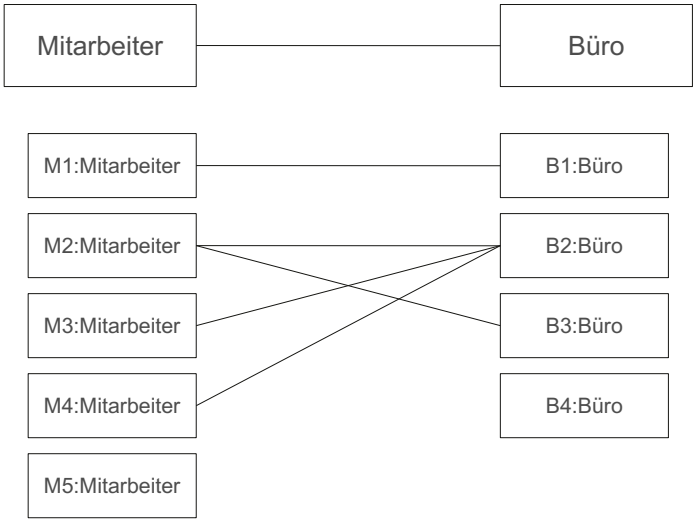
Multiplizität	Beschreibung
1	Genau eine Verbindung
0..1	Höchstens eine Verbindung
0..*	Beliebig viele Verbindungen
1..*	Mindestens eine Verbindung
n..m	Mindestens n höchstens m Verbindungen. Eher ungewöhnlich, nur zu nutzen wenn die Obergrenze zweifelsfrei feststeht, z.B. die Anzahl an Reifen an einem PKW hätte die Multiplizität 0..4. Häufig nutzt man in solchen Fällen dennoch die Multiplizität 0..*.

Assoziationen erhalten neben einem Namen auch Anzahlangaben (Multiplizitätsangaben). Dies gibt an mit wievielen Objekten der gegenüberliegenden Assoziationsseite je ein Objekt der Ausgangsseite verbunden ist.

Letztlich entscheiden diese Angaben, ob zum Verwalten der Kenntnisbeziehungen zwischen Objekten eine einfache Referenzvariable oder eine Collection über den Typ des Assoziationspartners genutzt werden muss.

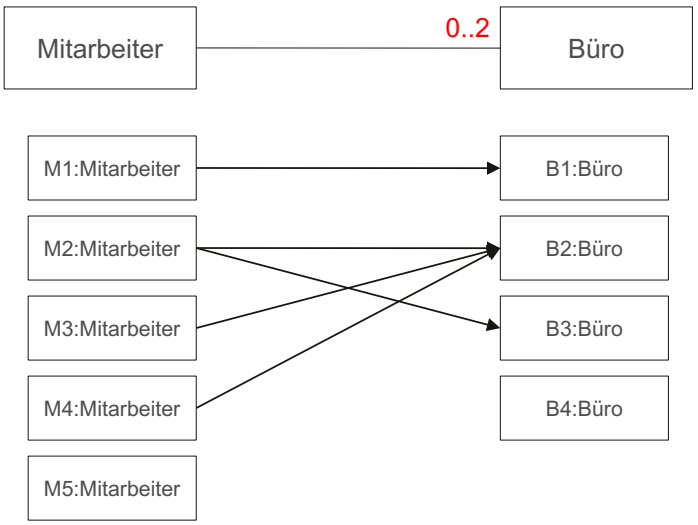
Multiplizitäten Beispiel

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



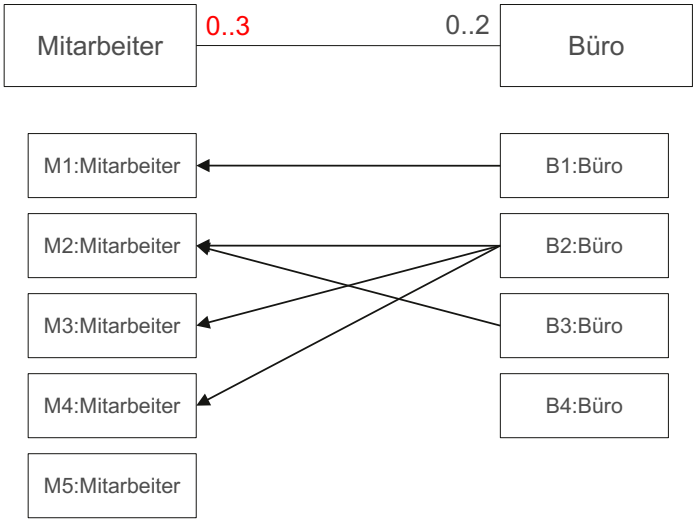
Multiplizitäten Beispiel
Sicht der Mitarbeiter

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



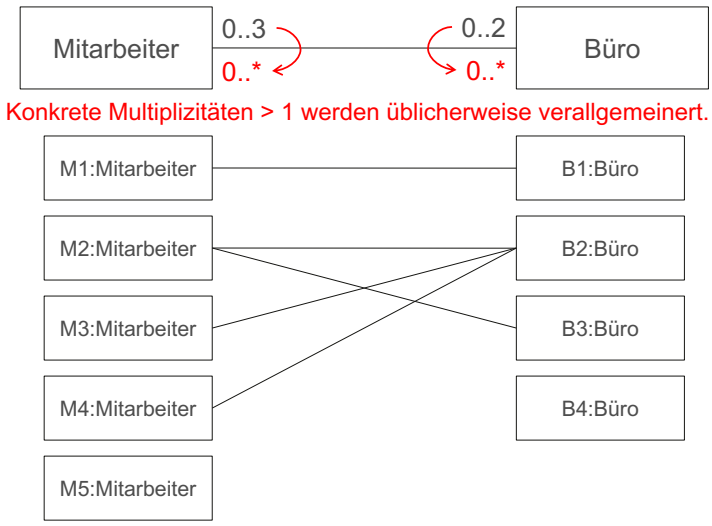
Multiplizitäten Beispiel
Sicht der Büros

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



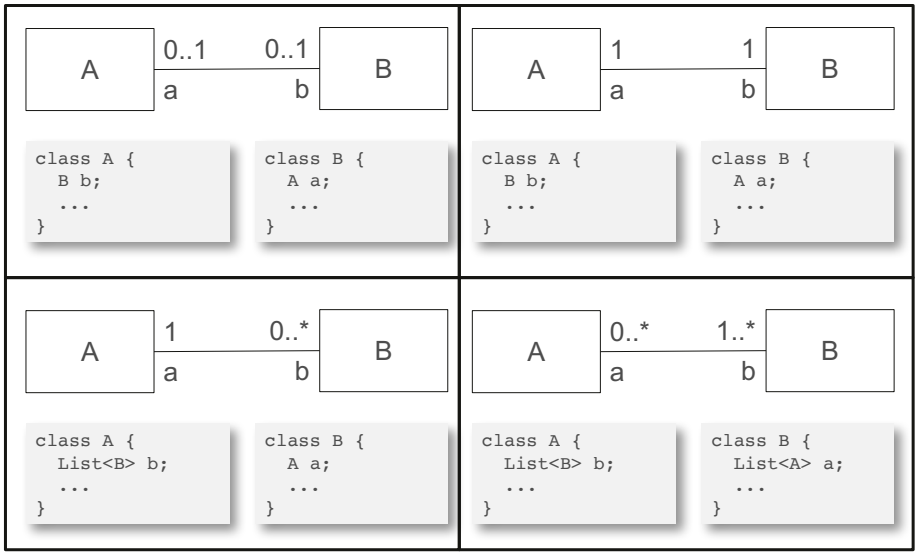
Multiplizitäten Beispiel
Angabe der Multiplizitäten

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



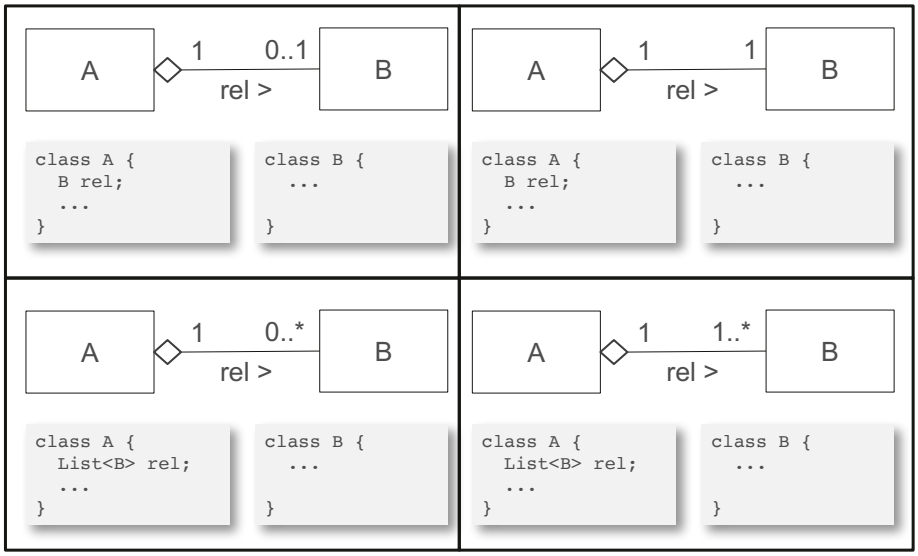
Transformationsregeln von Assoziationen

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



Transformationsregeln von Aggregationen/Kompositionen

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

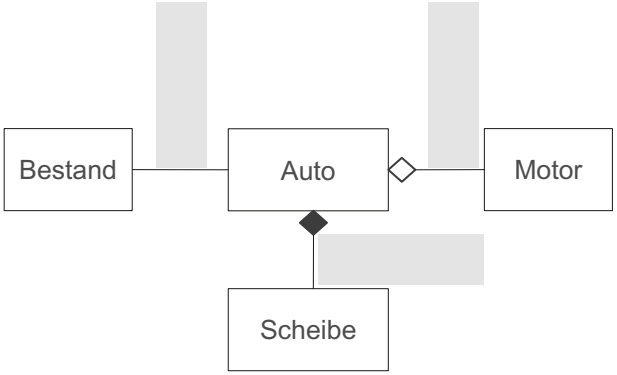


Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Gegeben ist folgendes UML Diagramm. Welche Arten von Kenntnisbeziehungen sind zwischen den Klassen definiert worden?

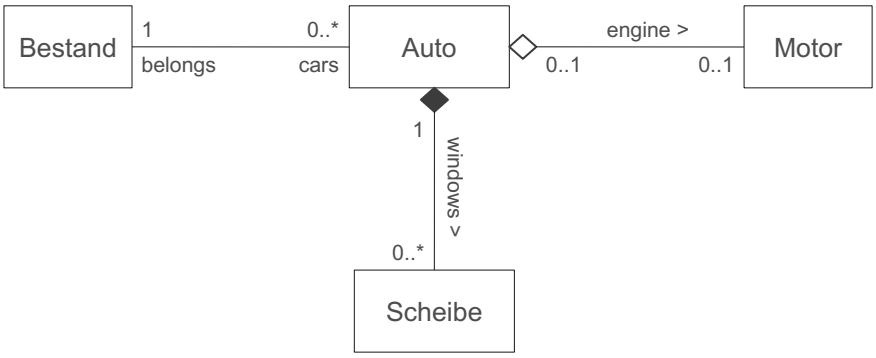


Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Das UML Diagramm ist wie folgt verfeinert worden. Welche Klassenrümpfe würden Sie daraus anhand der gezeigten Transformationsregeln ableiten?



Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Das UML Diagramm ist wie folgt verfeinert worden. Welche Klassenrümpfe würden Sie daraus anhand der gezeigten Transformationsregeln ableiten?

Lösung:

```
class Bestand {
  List<Auto> cars;
  ...

}
```

```
class Auto {
  Bestand belongs;
  Motor engine;
  List<Scheibe> windows;
  ...
}
```

```
class Motor {
  ...
  // bei Aggregationen
  // meist kein
  // Rückverweis
}
```

```
class Scheibe {
  ...
  // bei Kompositionen
  // meist kein
  // Rückverweis
}
```


Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Studierende sollen wie folgt angelegt und ausgegeben werden können.

```
Student s = new Student("Max", "Mustermann", 123456);
System.out.println(s);
```

```
Max Mustermann (MatrNr.: 123456)
```

Termine sollen wie folgt angelegt und ausgegeben werden können.

```
Termin t = new Termin(16, 15, 17, 45, "Übung VProg", "18-1.18");
System.out.println(t);
```

```
- 16:15h bis 17:45h : Übung VProg in 18-1.18
```

Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Studierenden können ferner Termine wie folgt zugeordnet werden.

```
Student t = new Student("Max", "Mustermann", 123456);
Termin t1 = new Termin(14, 30, 16, 00, "Vorlesung VProg", "18-0.01");
Termin t2 = new Termin(16, 15, 17, 45, "Übung VProg", "18-1.18");
s.insertTermin(t1);
s.insertTermin(t2);
```

Werden nun Studierende ausgegeben, sollen auch die Termine mit ausgegeben werden, die einem Studierenden zugeordnet sind.

```
System.out.println(s);
```

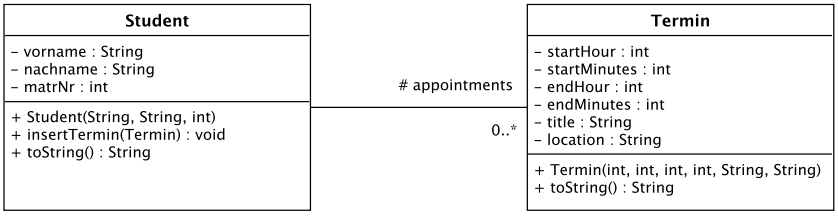
```
Max Mustermann (MatrNr.: 123456)
- 14:30h bis 16:00h : Vorlesung VProg in 18-0.01
- 16:15h bis 17:45h : Übung VProg in 18-1.18
```


Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Um sie zu unterstützen, ist ihnen folgendes UML-Diagramm gegeben.



Implementieren sie nun bitte `Student` und `Termin`.

Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Miniübung:



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Zusammenfassung

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- Grundsatz der Objektorientierung: Denken in Objekten
  - Klassen sind Baupläne
  - Objekte sind konkrete Ausprägungen dieser Baupläne
  - Objekte kommunizieren miteinander (Methoden) um ein Problem zu lösen

- Objekte kapseln ihre Daten (Information Hiding)
- Abstraktionsschritte in der Objektorientierung
- Objektkommunikation und Assoziationen
- Hierarchiebildung als Mittel der Kompexitätsbeherrschung
  - Vererbungshierarchien (is a-Hierarchien)
  - Zerlegungshierarchien (part of-Hierarchien)

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
85