Universität
Osnabrück

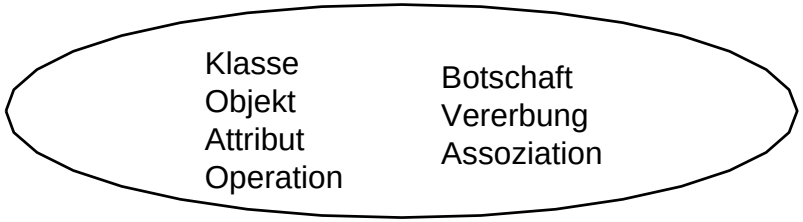
Informatik B – Objektorientierte Programmierung in Java

Vorlesung 01:
Objektorientierte Programmierung (Teil 1)

Inhalt

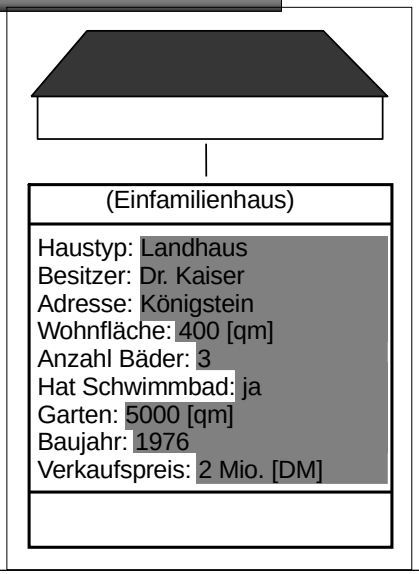
- Intuitive Einführung: Klassen und Objekte
- Klassen und Objekte
- Attribute
- Operationen
- Beispiel
- Überladen von Operationen

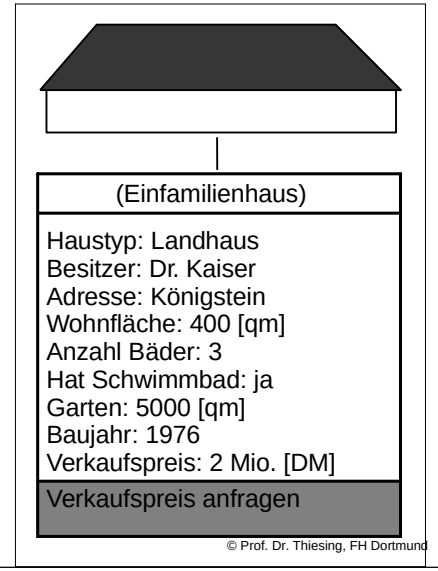
Intuitive Einführung



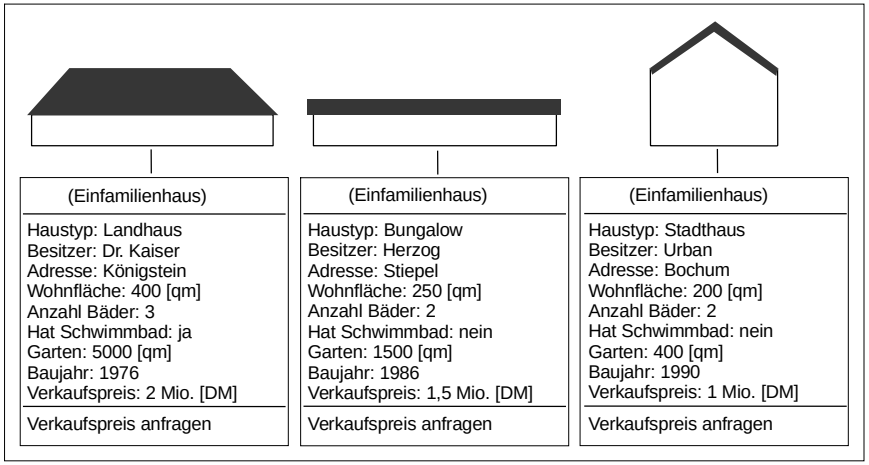
- Klasse: Beschreibung von Gegenständen mit gleichen Eigenschaften
- Objekt: Konkrete Instanz einer Klasse
- Attribut: Beschreibung der Objektdaten
- Operation: Ausführbare Tätigkeit (Funktion) eines Objektes

Intuitive Einführung



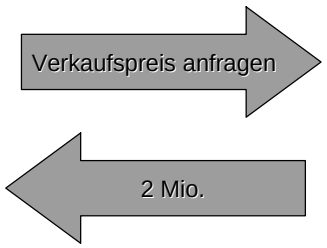


Intuitive Einführung



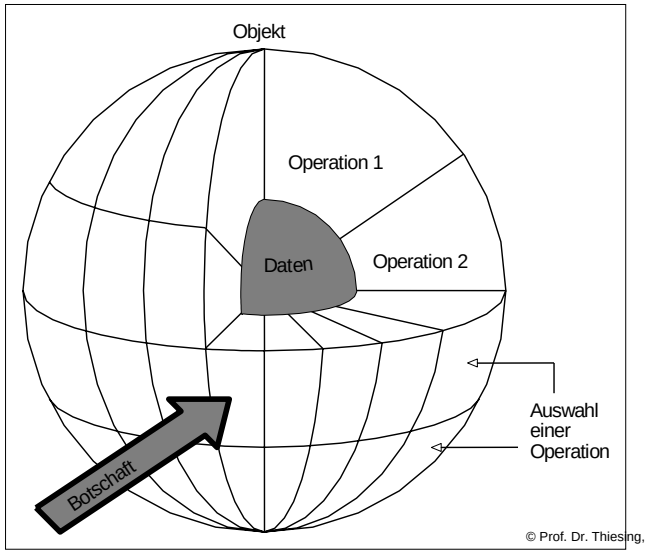
(Einfamilienhaus)	(Einfamilienhaus)	(Einfamilienhaus)
Haustyp: Landhaus Besitzer: Dr. Kaiser Adresse: Königstein Wohnfläche: 400 [qm] Anzahl Bäder: 3 Hat Schwimmbad: ja Garten: 5000 [qm] Baujahr: 1976 Verkaufspreis: 2 Mio. [DM]	Haustyp: Bungalow Besitzer: Herzog Adresse: Stiepel Wohnfläche: 250 [qm] Anzahl Bäder: 2 Hat Schwimmbad: nein Garten: 1500 [qm] Baujahr: 1986 Verkaufspreis: 1,5 Mio. [DM]	Haustyp: Stadthaus Besitzer: Urban Adresse: Bochum Wohnfläche: 200 [qm] Anzahl Bäder: 2 Hat Schwimmbad: nein Garten: 400 [qm] Baujahr: 1990 Verkaufspreis: 1 Mio. [DM]
Verkaufspreis anfragen	Verkaufspreis anfragen	Verkaufspreis anfragen

Intuitive Einführung

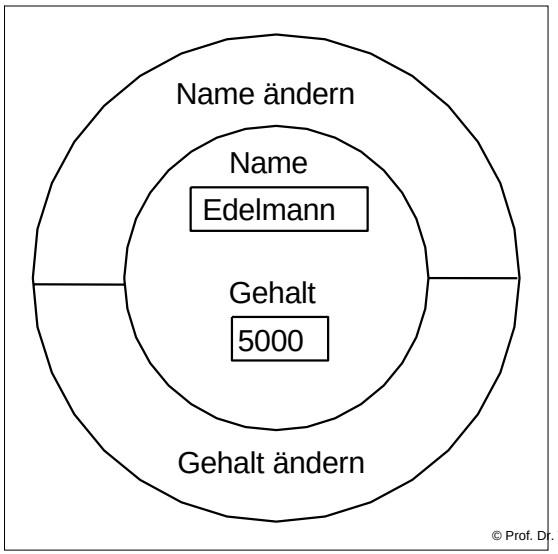


(Einfamilienhaus)
Haustyp: Landhaus Besitzer: Dr. Kaiser Adresse: Königstein Wohnfläche: 400 [qm] Anzahl Bäder: 3 Hat Schwimmbad: ja Garten: 5000 [qm] Baujahr: 1976 Verkaufspreis: 2 Mio. [DM]
Verkaufspreis anfragen

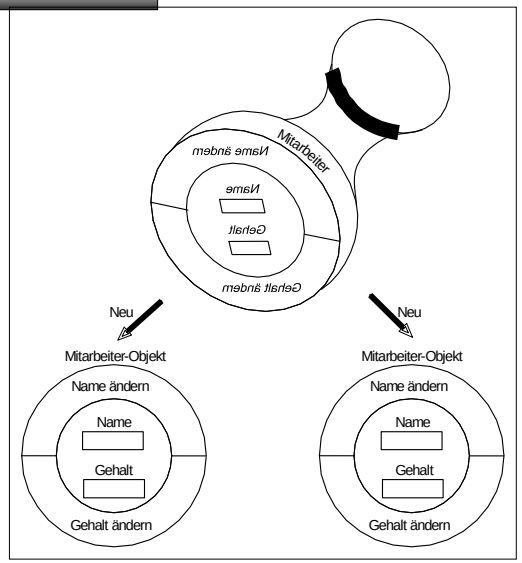
Intuitive Einführung



Intuitive Einführung



Intuitive Einführung



Klassen und Objekte

- Java-Programme bestehen aus Klassendefinitionen.
- Eine *Klasse* ist eine allgemeingültige Beschreibung von Dingen, die in verschiedenen Ausprägungen vorkommen können, aber alle eine gemeinsame Struktur und ein gemeinsames Verhalten haben.
- Sie ist ein Bauplan für die Erzeugung von einzelnen konkreten Ausprägungen.

Klassen und Objekte

- Diese Ausprägungen bezeichnet man als *Objekte* oder Instanzen der Klasse.
- Instanzvariablen, auch *Attribute* genannt, und *Operationen* (Methoden) sind die Bestandteile einer Klasse.
- Attribute beschreiben den Zustand und Operationen das Verhalten eines Objekts.

Klassen und Objekte

- Allgemeiner Aufbau einer Klasse in JAVA
  - Syntax

```
[Modifikator] class Klassenname
  [extends Basisklasse]
  [implements Schnittstellen]
 {
      Attributdeklarationen
      Operationsdeklarationen
 }
```

  - Die in eckigen Klammern angegebenen Teile sind optional und werden an späterer Stelle erklärt.

Klassen und Objekte

- Beispiel

```
class Auto
{
    String name;
    int erstzulassung;
    int leistung;
}
```

- Diese Klasse „**Auto**“ enthält keine Operationen, sondern lediglich die Variablen (Attribute) **name**, **erstzulassung** und **leistung**.

Klassen und Objekte

- Erzeugung eines Objekts
  - Um von einer Klasse ein Objekt anzulegen, muss eine Variable vom Typ der Klasse (sog. *Objektvariable*) deklariert und ihr mit Hilfe des **new**-Operators ein neu erzeugtes Objekt zugewiesen werden:

```
Auto meinKombi;
meinKombi = new Auto();
```

- Die erste Anweisung ist eine normale Variablendeklaration (wie z.B. **int a;**).
- Anstelle eines primitiven Typs wird hier der Name einer zuvor definierten Klasse verwendet.
- Im Unterschied zu einem primitiven Datentyp wird in der Variablen **meinKombi** nur eine Referenz gespeichert.

Klassen und Objekte

- Die zweite Anweisung erzeugt mit Hilfe des **new**-Operators ein neues Objekt der Klasse Auto und weist sie der Variablen **meinKombi** zu.
- In Java wird jede selbstdefinierte Klasse mit Hilfe des **new**-Operators instantiiert.
- Mit Ausnahme von Zeichenketten und Feldern, bei denen der Compiler auch Literale zur Objekterzeugung zur Verfügung stellt, gilt dies auch für alle vordefinierten Klassen der Java-Klassenbibliothek.
- Wie bei Variablen mit einem primitiven Datentyp lassen sich beide Anweisungen auch kombinieren. Das nachfolgende Beispiel deklariert und initialisiert die Variable **meinKombi**:

```
Auto meinKombi = new Auto();
```


Attribute

- Initialisierung
  - Attribute werden im Gegensatz zu lokalen Variablen mit dem Standardwert ihres Typs initialisiert.
  - Der Standardwert für Referenztypen ist **null**.

Attribute

- Zugriff auf Attribute
  - Der Zugriff auf ein Attribut erfolgt mit Hilfe der Punktnotation ***Objektname.Attributname***.
  - Um das Auto-Objekt in einen 250 PS starken Mercedes 600 des Baujahrs 1972 zu verwandeln, müssten folgende Anweisungen ausgeführt werden:

```
meinKombi.name = "Mercedes 600";
meinKombi.erstzulassung = 1972;
meinKombi.leistung = 250;
```


Attribute

- Ebenso wie der schreibende erfolgt auch der lesende Zugriff mit Hilfe der Punktnotation.
- Die Ausgabe des aktuellen Objektes auf dem Bildschirm könnte mit folgenden Anweisungen erledigt werden:

```
System.out.println("Name........: "+meinKombi.name);
System.out.println("Zugelassen..: "+meinKombi.erstzulassung);
System.out.println("Leistung....: "+meinKombi.leistung);
```


Operationen

- Operationen definieren das Verhalten von Objekten.
- Sie werden innerhalb einer Klassendefinition deklariert und haben Zugriff auf alle Daten des Objekts.
- Operationen sind das Pendant zu den Funktionen imperativer Programmiersprachen, arbeiten aber immer mit den Daten des aktuellen Objekts.

Operationen

- Globale Funktionen, die vollkommen unabhängig von einem Objekt oder einer Klasse existieren, gibt es in Java ebenso wenig wie globale Variablen.
- Sie werden später allerdings Klassenattribute und -operationen kennen lernen, die nicht an ein konkretes Objekt gebunden sind.

Operationen

- Syntax

```
[Modifikatoren]
Typ Operationsname([Parameterliste])
{
  // Anweisungen
}
```

  - Nach einer Reihe von optionalen Modifikatoren folgt der Typ des Rückgabewerts der Operation, ihr Name und eine optionale Parameterliste.
  - In geschweiften Klammern folgt dann der Rumpf, also die Liste der Anweisungen, die das Verhalten der Operation festlegen.

Operationen

- Beispiel
  - Die Erweiterung der Beispielklasse um eine Operation zur Berechnung des Alters eines Auto-Objekts würde beispielsweise so aussehen:

```
class Auto
{
    String name;
    int erstzulassung;
    int leistung;

    int alter()
    {
      return (2004 – erstzulassung);
    }
}
```


Operationen

- Aufruf
  - Der Aufruf einer Operation erfolgt ähnlich der Verwendung eines Attributes in Punktnotation.
  - Zur Unterscheidung von einem Variablenzugriff dienen die runden Klammern, die die (eventuell leere) Parameterliste umschließen.
  - Beispiel:
    - Das folgende Programm würde demnach die Zahl 14 auf dem Bildschirm ausgeben.

```
Auto golf1 = new Auto();
golf1.erstzulassung = 1990;
System.out.println(golf1.alter());
```


Operationen

- Die **`this`**-Referenz
  - Wie an der Definition von **`alter`** zu erkennen ist, darf eine Operation auf die Attribute ihrer Klasse zugreifen, ohne die Punktnotation zu verwenden.
  - Das funktioniert deshalb, weil der Compiler alle nicht in Punktnotation verwendeten Variablen **`x`**, die nicht lokale Variablen sind, auf das eigene Objekt (**`this)`** bezieht und damit als **`this.x`** interpretiert.

Operationen

- Die **this**-Referenz
  - Bei **this** handelt es sich um eine Referenz, der beim Anlegen eines Objekts automatisch erzeugt wird.
  - **this** ist eine Referenzvariable, die auf das aktuelle Objekt zeigt und dazu verwendet wird, die eigenen Operationen und Attribute anzusprechen.
  - Die **this**-Referenz ist immer verfügbar und kann wie eine ganz normale Objektvariable verwendet werden.
  - Er wird als versteckter Parameter an jede Operation übergeben, die keine Klassenoperation ist.

Operationen

- Die **this**-Referenz
  - Die Operation **alter** hätte also auch so geschrieben werden können:

```
int alter()
{
  return (2004 – this.erstzulassung);
}
```

    - Es ist manchmal sinnvoll, **this** explizit zu verwenden, wenn hervorgehoben werden soll, dass es sich um den Zugriff auf ein Attribut handelt.

Operationen

- Sichtbarkeit von Variablen
  - Innerhalb von Operationen verdecken lokale Variablen und formale Parameter die Attribute gleichen Namens.
  - Mit Hilfe von **this** kann zwischen einer lokalen Variablen und einem Attribut gleichen Namens unterschieden werden.
  - Beispiel

```
class Auto {
    String name;
    ...
    void setName(String name){
      this.name = name;
    }
}
```


Operationen

- Parameter
  - Eine Operation kann mit Parametern definiert werden.
  - Dazu wird bei der Definition eine Parameterliste innerhalb der Klammern angegeben.
  - Jeder formale Parameter besteht aus einem Typnamen und dem Namen des Parameters.
  - Soll mehr als ein Parameter definiert werden, so sind die einzelnen Definitionen durch Kommata zu trennen.

Operationen

- Parameter (call-by-value)
  - Alle Parameter werden in Java per *call-by-value* übergeben.
  - Beim Aufruf einer Operation wird also der aktuelle Wert in die Parametervariable kopiert und an die Operation übergeben.
  - Veränderungen der Parametervariablen innerhalb der Operation bleiben lokal und wirken sich nicht auf den Aufrufer aus.
  - Das folgende Beispiel definiert eine Operation **`printAlter`**, die das Alter des Autos insgesamt **`wieoft`** mal auf den Bildschirm ausgibt:

Operationen

- Parameter (call-by-value)
  - Beispiel

```
void printAlter(int wieoft)
{
   while(wieoft > 0)
   {
     wieoft--;
     System.out.println("Alter = "+alter());
   }
}
```

    - Obwohl der Parameter **`wieoft`** innerhalb der Operation verändert wird, merkt ein Aufrufer nichts von diesen Änderungen, da innerhalb der Operation mit einer Kopie (call-by-value) dieses primitiven Datentyps gearbeitet wird.

Operationen

- Parameter – primitive Datentypen (call-by-value)
  - Das folgende Programm würde das Alter des Objekts **`auto`** daher insgesamt neunmal auf dem Bildschirm ausgeben:

```
. . .
int a = 3;

auto.printAlter(a);
auto.printAlter(a);
auto.printAlter(a);

. . .
```


Operationen

- Parameter (nicht-primitive Datentypen)
  - Objekte (und Felder) werden als Parameter ebenfalls per call-by-value an eine Operation übergeben.
  - Da aber nur die Referenz kopiert wird, steht innerhalb der Operation ein Verweis auf das Originalobjekt zur Verfügung (wenn auch nur in kopierter Form).
  - Veränderungen an dem Objekt sind somit natürlich auch für den Aufrufer der Operation sichtbar (*Seiteneffekt*).

Operationen

- Parameter (nicht-primitive Datentypen)
  - Die Übergabe von Objekten an Operationen hat damit zwei wichtige Konsequenzen:
    - Die Operation erhält keine Kopie, sondern arbeitet mit dem Originalobjekt.
    - Die Übergabe von Objekten ist performant, gleichgültig wie groß sie sind.
  - Sollen Objekte kopiert werden, so muss dies explizit durch Aufruf der Operation **`clone`** erfolgen.

Operationen

- Rückgabewert
  - Der Typ einer Operation wird zum Zeitpunkt der Deklaration festgelegt und bestimmt den Typ des Rückgabewerts.
  - Dieser kann von einem beliebigen primitiven Typ, einem Referenztyp oder vom Typ **`void`** sein.
  - Die Operationen vom Typ **`void`** haben gar keinen Rückgabewert und dürfen nicht in Ausdrücken verwendet werden.
  - Sie sind lediglich wegen ihrer Seiteneffekte von Interesse und dürfen daher nur als Ausdrucksanweisung verwendet werden.

Operationen

- Rückgabewert
  - Die **`return`**-Anweisung
    - Hat eine Operation einen Rückgabewert (ist also nicht vom Typ **`void`**), so muss sie mit Hilfe der **`return`**-Anweisung einen Wert an den Aufrufer zurückgeben.
    - Die **`return`**-Anweisung hat folgende Syntax:

      `return` ***`Ausdruck`*****`;`**
    - Wenn diese Anweisung ausgeführt wird, führt dies zum Beenden der Operation, und der Wert des angegebenen Ausdrucks wird an den Aufrufer zurückgegeben.
    - Der Ausdruck muss dabei zuweisungskompatibel zum Typ der Operation sein.

Beispiel

- Die selbstdefinierten Klassen **`Auto`** und **`Autoverwaltung`**

Überladen von Operationen

- In Java ist es erlaubt, Operationen zu überladen, d.h. innerhalb einer Klasse zwei unterschiedliche Operationen mit demselben Namen zu definieren.
- Der Compiler unterscheidet die verschiedenen Varianten anhand der Anzahl, des Typs und der Position der Parameter.
- Es ist also nicht erlaubt, zwei Operationen mit exakt demselben Namen und identischer Parameterliste zu definieren.

Überladen von Operationen

- Dabei werden auch zwei Operationen, die sich nur durch den Typ ihres Rückgabewertes unterscheiden, als gleich angesehen.
- Das Überladen von Operationen ist dann sinnvoll, wenn die gleichnamigen Operationen auch eine vergleichbare Funktionalität haben.
- Eine typische Anwendung von überladenen Operationen besteht in der Simulation von variablen Parameterlisten (die als Feature direkt in Java nicht zur Verfügung stehen).

Überladen von Operationen

- Auch, um eine Operation, die bereits an vielen verschiedenen Stellen im Programm aufgerufen wird, um einen weiteren Parameter zu erweitern, ist es nützlich, diese Operation zu überladen, um nicht alle Aufrufstellen anpassen zu müssen.
- Das folgende Beispiel erweitert die Klasse Auto um eine weitere Operation **`alter`**, die das Alter des Auto nicht nur zurückgibt, sondern es auch mit einem als Parameter übergebenen Titel versieht und auf dem Bildschirm ausgibt:

Überladen von Operationen

- Beispiel:

```
class Auto
{  String name;
   int erstzulassung;
   int leistung;

   int alter()
   { return (2004 – erstzulassung);
   }

   int alter(String titel)
   { int alter = alter();
     System.out.println(titel+alter);
     return alter;
   }
}
```

Überladen von Operationen

- Signatur einer Operation
  - Innerhalb dieser Operation wird der Name **`alter`** in drei verschiedenen Bedeutungen verwendet:
    - **`alter`** ist der Name der Operation selbst.
    - Die lokale Variable **`alter`** wird definiert.
    - Die parameterlose **`alter`**-Operation wird aufgerufen.
  - Der Compiler kann die Namen in allen drei Fällen unterscheiden, denn er arbeitet mit der Signatur der Operation.

Überladen von Operationen

- Signatur einer Operation
  - Unter der *Signatur* einer Operation versteht man ihren internen Namen.
  - Dieser setzt sich aus dem nach außen sichtbaren Namen plus codierter Information über die Reihenfolge und Typen der formalen Parameter zusammen.
  - Die Signaturen zweier gleichnamiger Operationen sind also immer dann unterscheidbar, wenn sie sich wenigstens in einem Parameter voneinander unterscheiden.