

# Grundlagen der Informatik für Ingenieure I

## 3 Einführung in das objektorientierte Programmier-Paradigma

---

- 3.1 *Software Design-Methoden*
  - 3.1.1 *Top-down structured design*
  - 3.1.2 *Data-driven design*
  - 3.1.3 *Object-oriented design*
- 3.2 *Historische Entwicklung*
- 3.3 *OO-Programmierung - Grundlagen*
  - 3.3.1 *Klassen, Objekte, Methoden*
  - 3.3.2 *Eigenschaften (Attribute) und Verhalten (Methoden)*
  - 3.3.3 *Kreieren einer Klasse*
  - 3.3.4 *Beispiel: Motorbike*

### 3.1 Software Design-Methoden

---

- ◆ ***Top-down structured design***
  - Traditionelle Software-Design-Methode
- ◆ ***Data-driven design***
  - Orientiert sich an der Abbildung von Eingabedaten auf Ausgabedaten
- ◆ ***Object-oriented design***
  - die bisher "modernste" Methode
  - erste zugehörige Programmiersprache: SIMULA (1967)

# 1 Top-down structured design

- ◆ Wesentlich durch die traditionellen Programmiersprachen (Fortran, Cobol, C, ...) beeinflusst.
- ◆ Einheit für Dekomposition: **Unterprogramm**
  - Resultierendes Programm hat die Form eines Baumes, in dem Unterprogramme ihre Aufgaben durch den Aufruf weiterer Unterprogramme erledigen.
- ◆ Algorithmische Dekomposition zur Zerlegung größerer Probleme
  - Zentrale Sicht:
    - Ausführung einer Problemlösung zerlegt in Teilproblemlösungen und Einzelschritten;
    - durch Vorgabe einer Reihenfolge --> Gesamtlösung

# 1 Top-down structured design

- ◆ Eignung für die Strukturierung heutiger, sehr großer Softwaresysteme wird bezweifelt, trotzdem existieren große Softwaresysteme auf der Basis dieser Methode.
- ◆ Diese Methode erfaßt nicht explizit:
  - Datenabstraktion & *Information Hiding*
  - Nebenläufigkeit
- ◆ Teilproblemlösungen operieren auf globalem Datenbestand, lokale Daten sind im allgemeinen Zwischenspeicher.
- ◆ Bislang am häufigsten eingesetzte Design-Methode

## 2 Data-driven design

---

- ◆ Grundlegende Arbeiten u. a. von Jackson
- ◆ Softwarestruktur beruht auf Abbildung von Daten-Eingabe in Daten-Ausgabe
  - Anwendung vor allem im Bereich *Information Management*
- ◆ Probleme vor allem mit zeitkritischen Ereignissen
- ◆ hat sich nicht durchsetzen können.

## 3 Object-oriented design

---

Bertrand Meyer:

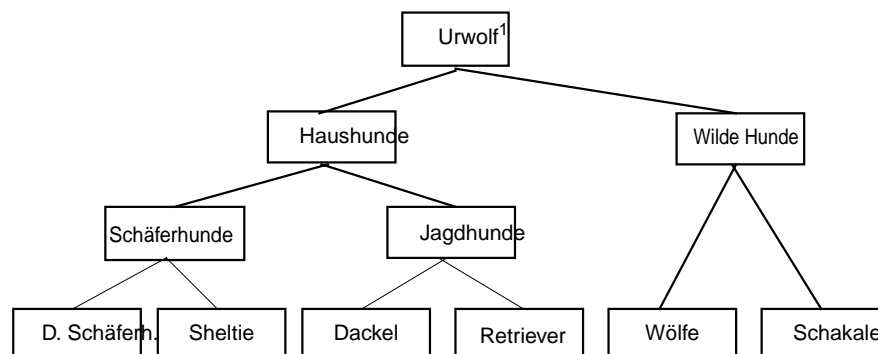
*"Rechner führen Operationen auf bestimmten Objekten aus. Um flexiblere und wiederverwendbare Systeme zu erhalten, ist es daher sinnvoller, die Software-Struktur an diesen Objekten statt an den Operationen zu orientieren."*

### 3 Object-oriented design

- ◆ Softwaresystem wird als Sammlung kooperierender **Objekte** modelliert
- ◆ Objekte (Instanzen) werden aus dem template (= Muster) "**Klasse**" erzeugt.
- ◆ Eine Klasse ist die Beschreibung der Eigenschaften aller aus dieser Klasse erzeugten Objekte.
- ◆ **Hierarchie** entsteht durch **Oberklassen** und **Unterklassen**, wobei jeweils die Unterklasse eine Spezialisierung der Oberklasse ist.

### 3 Object-oriented design

- ◆ Klassenhierarchie (verbreitet in der Wissenschaft):



1) Biologen mögen mir diese Taxonomie nachsehen

## 3 Object-oriented design

- ◆ Wird von modernen, höheren Programmiersprachen unterstützt:
  - Smalltalk, C++, ADA, Java
- ◆ Grundlage: objektorientierte Dekomposition
- ◆ Vorteile:
  - Wiederverwendung gemeinsamer Mechanismen;
    - Oberklassen beschreiben gemeinsame Eigenschaften, die von Unterklassen übernommen werden;
    - Software wird kompakter
  - Software ist leichter zu ändern und weiterzuentwickeln
    - Änderungen sind örtlich begrenzt;
    - Seiteneffekte werden vermieden;
    - Änderungen einer Oberklasse haben eine einheitliche Auswirkung auf die Unterklassen

## 3.2 Historische Entwicklung

- **Generationen von Programmiersprachen:**
  - ◆ **Erste** Generation (1954 - 1958)
    - Mathematische Ausdrücke (FORTRAN I, ALGOL58)
  - ◆ **Zweite** Generation (1959 - 1961)
    - Unterprogramme, getrennte Übersetzung (FORTRAN II)
    - Blockstruktur, Datentypen (ALGOL60)
    - Datenbeschreibung, Dateibehandlung (COBOL)
    - Listenverarbeitung, Zeiger (Lisp)

## 3.2 Historische Entwicklung

- **Generationen von Programmiersprachen (cont):**
  - ◆ **Dritte** Generation (1962 - 1970)
    - verschiedene ALGOL-Nachfolger (ALGOL68, Pascal)
    - Klassen, Datenabstraktion (Simula)
  - ◆ **Vierte** Generation (1970 - 1995)
    - Viele neue Sprachen
    - nur wenige haben Bedeutung erlangt:
      - C, Pascal, Modula, Smalltalk, C++, Java

## 3.3 OO-Programmierung - Grundlagen

- **Definition**
  - ◆ **OO** ist eine Methode der Implementierung.
  - ◆ Programme sind in Form von Mengen kooperierender **Objekte**, die in den zugehörigen Klassen beschrieben werden, organisiert.
  - ◆ Ein Objekt wird aus einer **Klasse** erzeugt.
  - ◆ Die Klassen sind Bestandteil einer über **Vererbungsbeziehungen** definierten **Hierarchie** von Klassen.

# 1 Klassen, Objekte, Methoden

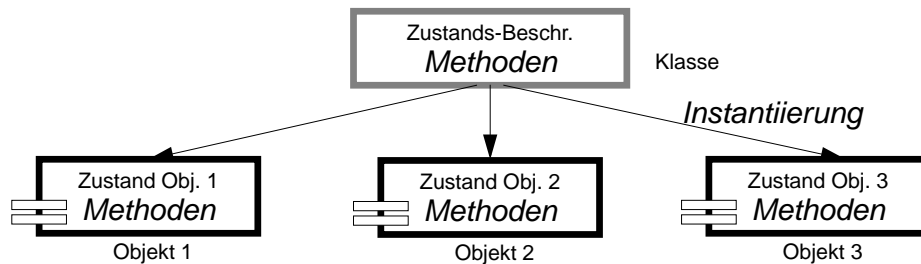
- ◆ **Klasse (Class)** = Objektbeschreibung (Template; Muster; Vorschrift)
  - Beschreibung der Eigenschaften (→Attribute) der aus dieser Klasse erzeugten Objekte.
  - und Beschreibung der Operationen der Objekte (→**Methoden (methods)**) dieser Klasse.
  - **Vererbung (inheritance)**: Relation zwischen Klassen.
  - Objektorientierte Programmierung (OOP) verwendet Objekte und nicht Algorithmen als die grundlegenden Bausteine der Problemlösung.
  - Die Begriffe Objekt und Instanz werden synonym verwendet.

# 1 Klassen, Objekte, Methoden

- ◆ **Instantiierung** = Erzeugung eines Objekts (einer Instanz) einer Klasse.
  - Alle Objekte können den in der Klasse implementierten Code der Methoden gemeinsam nutzen.
  - Sie haben aber jeweils eine eigene Version des Objektzustandes (Datenbasis), gemäß der Strukturbeschreibung in der Klasse.
- ◆ **Objektvariablen (Attribute)** = Variablen eines Objekts (einer Instanz)
  - = Zustand

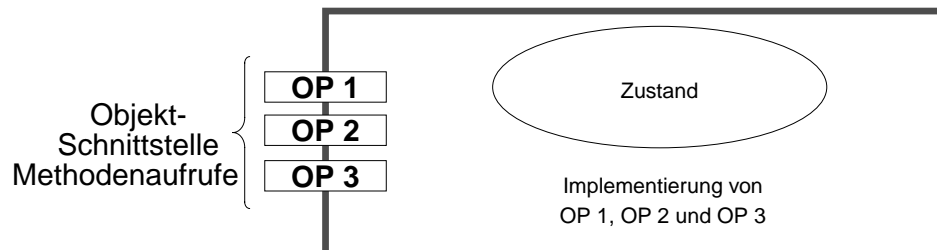
# 1 Klassen, Objekte, Methoden

- ◆ **Methoden** = Operationen eines Objekts (Schnittstelle)
  - Die Methoden eines Objekts definieren das Verhalten (die Zustandsübergänge) eines Objekts.
- ◆ Graphische Darstellung:



# 1 Klassen, Objekte, Methoden

- ◆ **Objekt** = Variablen (Zustand) + Operationen



- Zusammenfassung von Datenstrukturen und Operationen auf diese Datenstrukturen zu einer Programmeinheit.
- ◆ Klassisches Beispiel: **Warteschlange**
  - Warteschlange ist eine Ansammlung von Elementen mit
  - der Variablen **queueLength** (Warteschlangenlänge)
  - und den Operationen **append** (anhängen) und **remove** (entfernen)



## 2 Eigenschaften (Attribute) und Verhalten (Methoden)

◆ Beispiel: "abstraktes" Motorrad:

- Zunächst führen wir eine Klasse **Motorbike** ein.
- Diese Klasse hat einige Eigenschaften (Attribute).
- Diese Attribute werden mit Hilfe von → Variablen und deren → Wertebereiche beschrieben, wie z. B.:

```
make: [Honda, BMW, Yamaha]
color: [red, green, silver, brown]
engineOn: [true, false]
[...]: Wertebereich
```

- *color, make, engineOn* sind die Namen der Variablen (Objektvariablen).
- Der Inhalt der Objektvariablen beschreibt den Zustand des Objekts.
- z.B.: make = BMW, color = green, engineOn = false

## 2 Eigenschaften (Attribute) und Verhalten (Methoden)

- Das Verhalten dieser Klasse wird von Methoden beschrieben.
- Diese Methoden können die Inhalte der Variablen verändern.
- d. h., den Zustand des Objekts:

```
Start the engine    --> startEngine
Stop the engine     --> stopEngine
Show Attributes     --> showAtts
```

## 3 Kreieren einer Klasse

- ◆ Definition einer Klasse:

```
class Motorbike {
}
```

- ◆ Vereinbarung der Objektvariablen (Attribute):

```
String make;
String color;
boolean engineOn;
```

- ◆ Definition einer Methode:

```
void startEngine() {
    if ( engineOn == true )
        System.out.println( "The engine is already on." );
    else {
        engineOn = true;
        System.out.println( "The engine is now on." );
    }
}
```

## 3 Kreieren einer Klasse

- ◆ Der bisher erstellte Programmcode kann aber noch nicht ausgeführt werden.
- ◆ Was fehlt, ist eine Methode, die die Klasse **Motorbike** "benutzt".
- ◆ Der Java-Interpreter muß wissen, mit welcher Anweisung er die Programmausführung beginnen soll.
- ◆ Deshalb muß es **genau eine Methode** geben in der ein Objekt der Klasse **Motorbike** erzeugt wird und dessen Methoden dort benutzt werden.
- ◆ Diese Methode besitzt den Namen "**main**"!

## 3 Kreieren einer Klasse

### ◆ Methode: `main`

```
public static void main( String args[] ) {
    Motorbike myBike;

    myBike = new Motorbike();

    myBike.setMake( "Yamaha" );
    myBike.setColor( "yellow" );

    System.out.println( "Calling showAtts..." );
    myBike.showAtts();

    System.out.println( "-----" );

    System.out.println( "Starting engine..." );
    myBike.startEngine();

    .....
}
```

- `myBike` ist Objekt der Klasse **Motorbike** (wird mit **"new"** erzeugt).
- `myBike.startEngine()` ruft die Methode `startEngine()` für das Objekt `myBike` auf.

## Programm:

```
class Motorbike {
    private String make;
    private String color;
    private boolean engineOn = false;

    void startEngine() {
        if ( engineOn == true )
            System.out.println( "The engine is already on." );
        else {
            engineOn = true;
            System.out.println( "The engine is now on." );
        }
    }

    void showAtts() {
        System.out.println( "This motorbike is a " + color + " " + make );
        if ( engineOn == true )
            System.out.println( "The engine is on." );
        else
            System.out.println( "The engine is off." );
    }

    void setMake( String comp ) {
        make = comp;
    }

    void setColor( String co ) {
        color = co;
    }
}
```

## Programm (Fortsetzung):

```

public static void main( String args[] ) {
    Motorbike myBike;

    myBike = new Motorbike();

    myBike.setMake( "Yamaha" );
    myBike.setColor( "yellow" );

    System.out.println( "Calling showAtts..." );
    myBike.showAtts();

    System.out.println( "-----" );

    System.out.println( "Starting engine..." );
    myBike.startEngine();

    System.out.println( "-----" );

    System.out.println( "Calling showAtts..." );
    myBike.showAtts();

    System.out.println( "-----" );

    System.out.println( "Starting engine..." );
    myBike.startEngine();
}
}

```

## Ergebnis

```

fau40u - home/inf/bolch/Lehrveranstaltungen/GDI-MASCH/GDI2003/Vorlesung - Konsole <3>
File Sessions Settings Help
fau40u: 17:43 Beispiele/Kap03 [17] : xemacs Motorbike.java &
[1] 12478
fau40u: 17:43 Beispiele/Kap03 [18] : ls
Motorbike.java
fau40u: 17:43 Beispiele/Kap03 [19] : javac Motorbike.java
fau40u: 17:44 Beispiele/Kap03 [20] : ls
Motorbike.class Motorbike.java
fau40u: 17:44 Beispiele/Kap03 [21] : java Motorbike
Calling showAtts...
This motorbike is a yellow Yamaha
The engine is off.
-----
Starting engine...
The engine is now on.
-----
Calling showAtts...
This motorbike is a yellow Yamaha
The engine is on.
-----
Starting engine...
The engine is already on.
fau40u: 17:44 Beispiele/Kap03 [22] :

```