Grundlagen der Programmierung

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorlesungsskript zum Sommersemester 2020

6. Vorlesung (25. Mai 2020)

Dr. Jin Gerlach
Christian Olt
Fachgebiet Wirtschaftsinformatik | Software & Digital Business
Fachbereich Rechts- und Wirtschaftswissenschaften
Technische Universität Darmstadt

WIRTSCHAFTS INFORMATIK

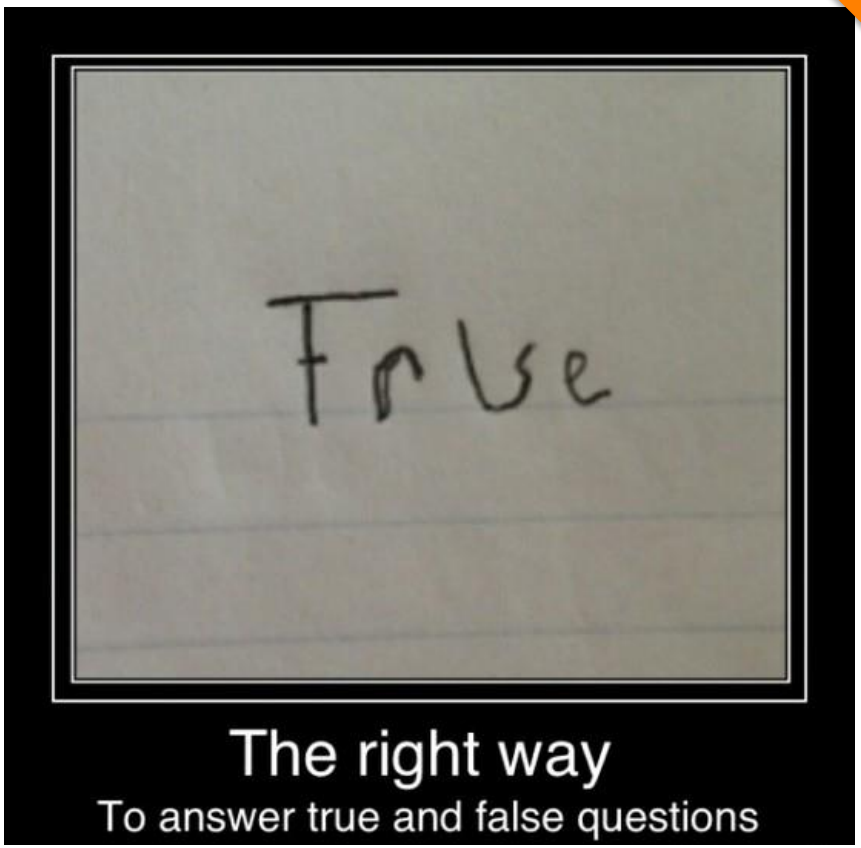
True oder false?

1. Die Variablen der Parameterliste einer Methode können innerhalb des Anweisungsblocks der Methode verwendet werden

2. Für Methoden mit Rückgabetyp „boolean“ ist die Verwendung des „return“-Statements optional

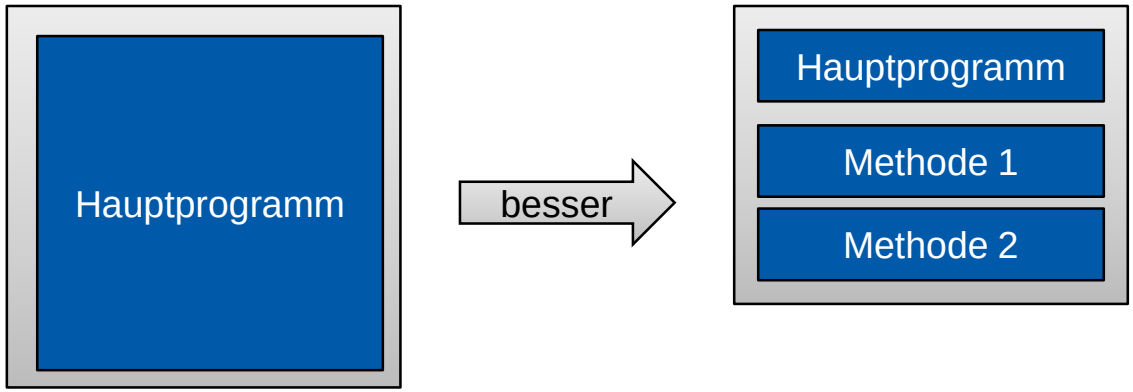
3. Überladene Methoden besitzen denselben Bezeichner aber unterschiedliche Parameterlisten

4. Durch „Casting“ können in Java Datentypen ineinander umgewandelt werden



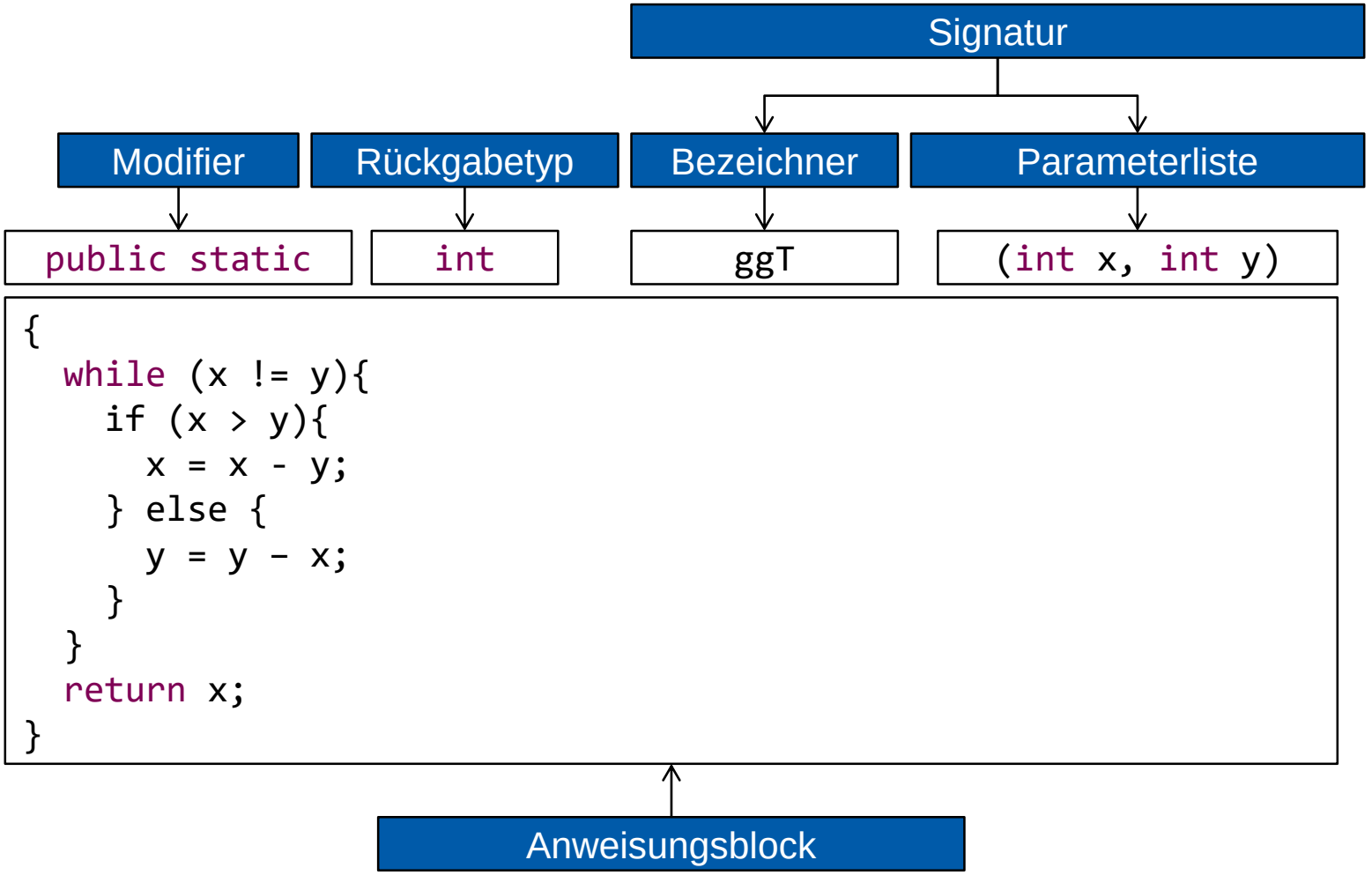
Methoden

- Methoden sind **Unterprogramme**
- Sie stellen (oftmals häufig wiederkehrende) **Hilfsfunktionen** bereit, die somit nur einmal implementiert, aber beliebig oft durch das Hauptprogramm aufgerufen werden können



- Wir haben bereits Methoden benutzt, z. B. `println(...)`

Elemente einer Methode



Überladene Methoden

- Eine Methode heißt **überladen**, wenn in derselben Klasse gleichnamige Methoden mit unterschiedlicher Signatur existieren (*Overloading*)
- Die überladenen Methoden können unterschiedliche Implementierungen enthalten
- Auswahl der auszuführenden Methode erfolgt anhand der **Signatur**
- Beispiel:

```
public static double average(double a, double b) {
  return (a + b) / 2;
}

public static double average(double a, double b, double c) {
  return (a + b + c) / 3;
}
```

Typenumwandlung („Casting“)

- Zwar ist Java eine „getypte“ Sprache (d.h. es werden Datentypen verwendet) aber Datentypen können bei Bedarf konvertiert werden (dies wird als „**Casting**“ bezeichnet)
- Beispiel:

```
int i = 5;
double d = i;
```

- Java unterscheidet zwei Arten des Castings:
  - **implizite** (automatische) Typumwandlung
  - **explizite** (manuelle) Typumwandlung

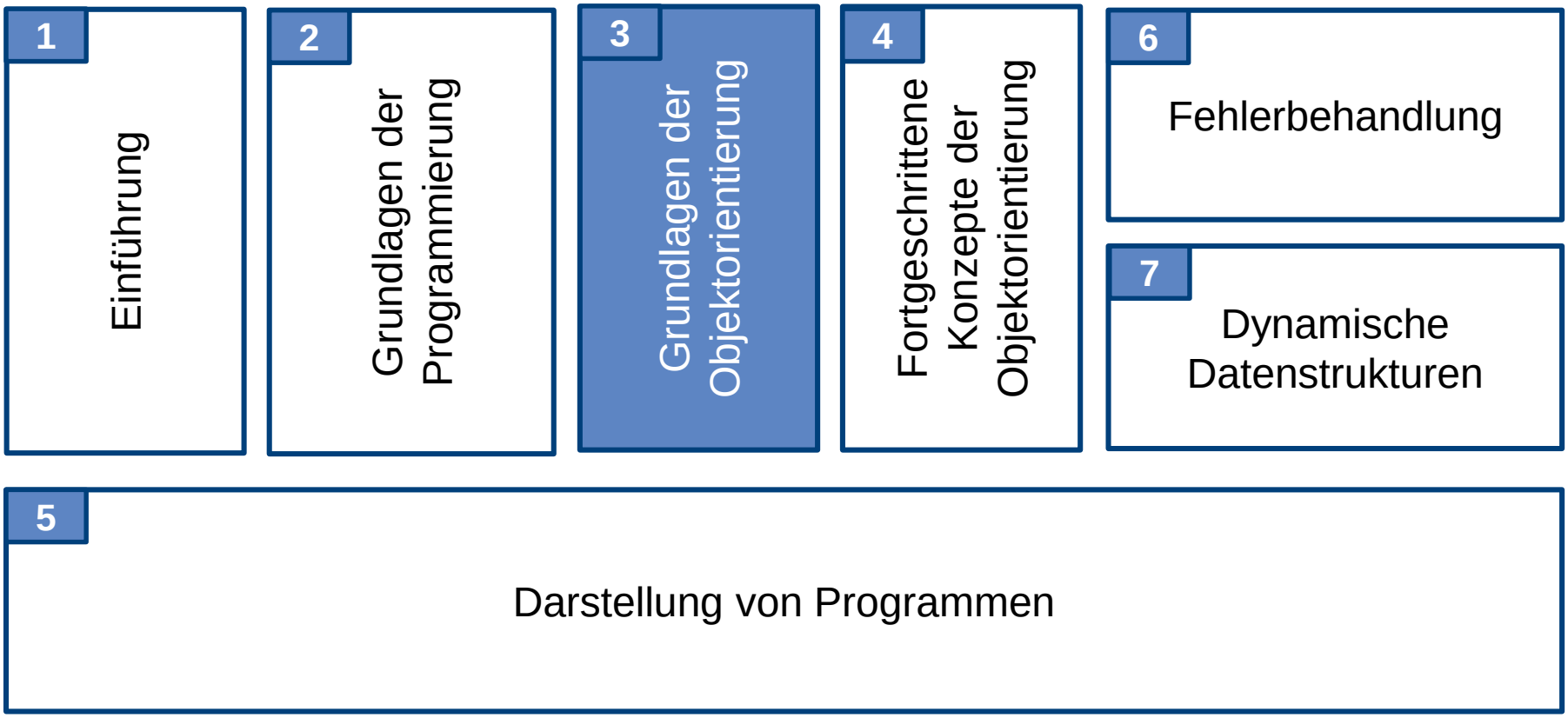
Abstrakte Lernziele

Die Studierenden sind nach den Veranstaltungen in der Lage,

- **Grundbegriffe der Programmierung** zu kennen,
- **Operatoren, Daten- und Kontrollstrukturen** zu kennen, um einfache Aufgabenstellungen algorithmisch lösen zu können,
- Grundlagen der **Objektorientierten Programmierung** kennen und anwenden,
- **Fortgeschrittene Konzepte der Objektorientierung** kennen (und anwenden),
- einfache **Java-Programme** zu lesen und zu schreiben und
- Programme mit **UML-Diagrammen** zu modellieren.

Thematische Übersicht

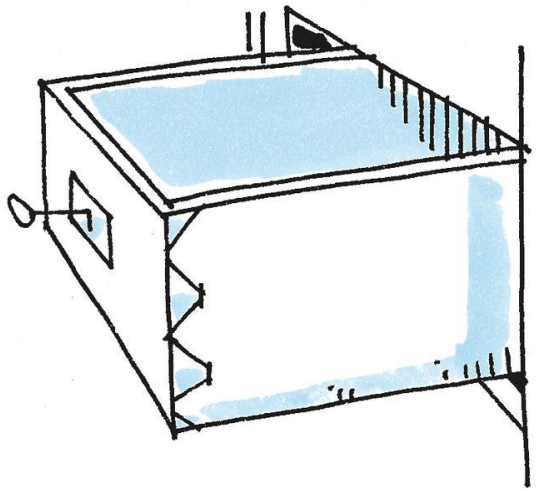
TECHNISCHE UNIVERSITÄT DARMSTADT



WIRTSCHAFTS INFORMATIK

Variablen und Arten von Datentypen

- **Variablen** sind Behälter für genau einen Wert und es gilt:
  - Variablen haben einen **Datentyp** (z. B. `int`)
  - Variablen haben einen **Bezeichner** (z. B. `x` oder `besteVariableDerWelt`)
  - Variablen haben einen **Wert** (z. B. `42`)



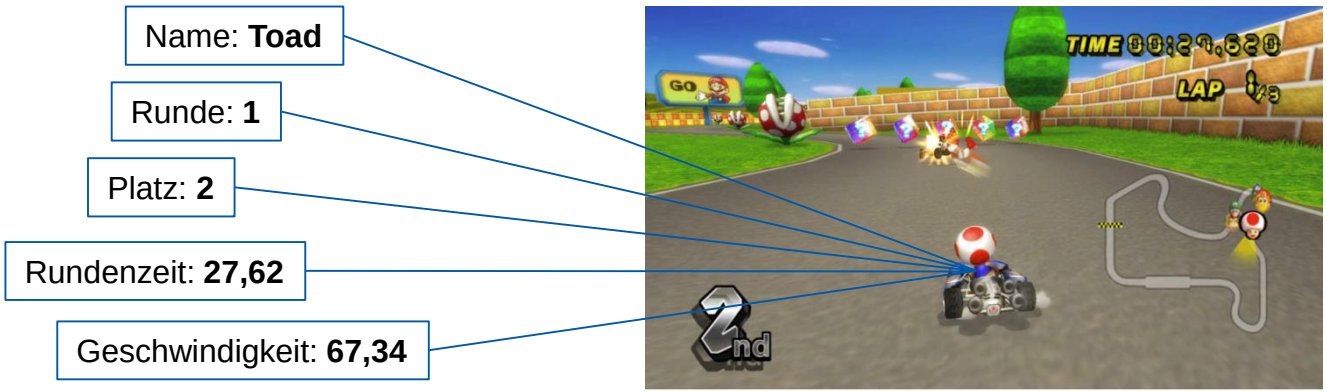
- Wir unterscheiden **zwei Arten von Datentypen**:
  - **Einfache** Datentypen:
    - Wahrheitswert (`boolean`) → true, false
    - Einzelzeichen (`char`) → z. B.: `'a'`, `'b'`, `'1'`, `'2'`, `'%'`, `'_'`
    - Numerische Datentypen
      - Ganzzahlige Datentypen (`byte`, `short`, `int`, `long`) → z. B. 123
      - Gleitkommatypen (`float`, `double`) → z. B. 1.25
  - **Strukturierte** Datentypen (auch: Referenz-Datentypen)

Notwendigkeit „komplexerer“ Datentypen

TECHNISCHE UNIVERSITÄT DARMSTADT

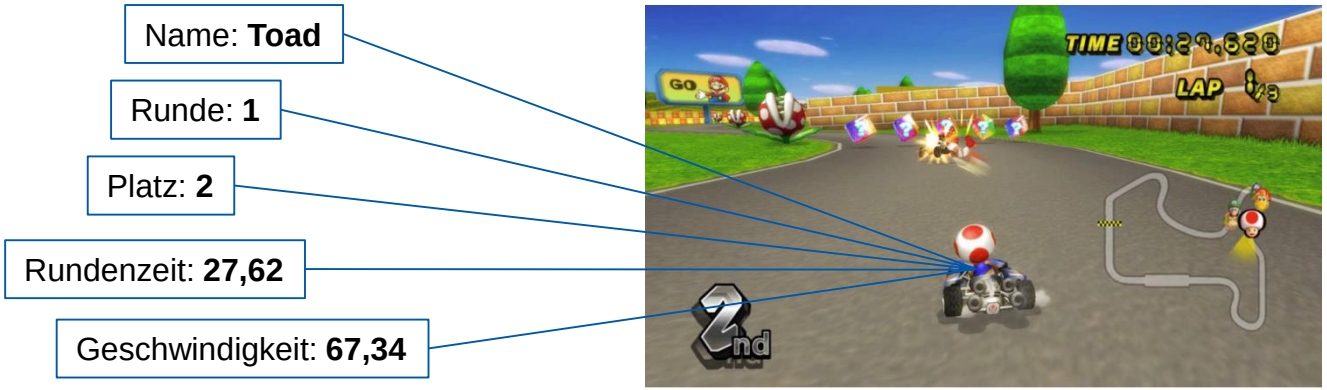
- Mit einfachen Datentypen können einzelne **Zahlen, Buchstaben oder Wahrheitswerte** gespeichert werden
- Arrays bieten eine Möglichkeit, **mehrere Werte des gleichen Typs** zusammen zu speichern

- Oft ist es aber gewünscht, **Werte verschiedenen Typs als „logische Einheit“** in einem Programm zu verwalten



WIRTSCHAFTS INFORMATIK

Definition eigener Datentypen durch Klassen



Klassen erlauben es, eigene strukturierte Datentypen zu definieren:

```
public class Fahrer {

  public String name;
  public int runde;
  public int platz;
  public double rundenzeit;
  public double geschwindigkeit;

}
```

```
public class MarioKart {
  public static void main(String[] args){
    Fahrer einFahrer = new Fahrer();
    einFahrer.name = "Toad";
    einFahrer.runde = 1;
    einFahrer.platz = 2;
  }
}
```

Ergänzung durch Verhalten

- Häufig zeichnen sich diese strukturierten Datentypen durch ein bestimmtes **Verhalten** aus
- z.B. kann ein Fahrer beschleunigen, bremsen, eine Runde beenden...
- Mit Hilfe von Methoden kann auch das Verhalten des oben modellierten Datentyps festgelegt werden

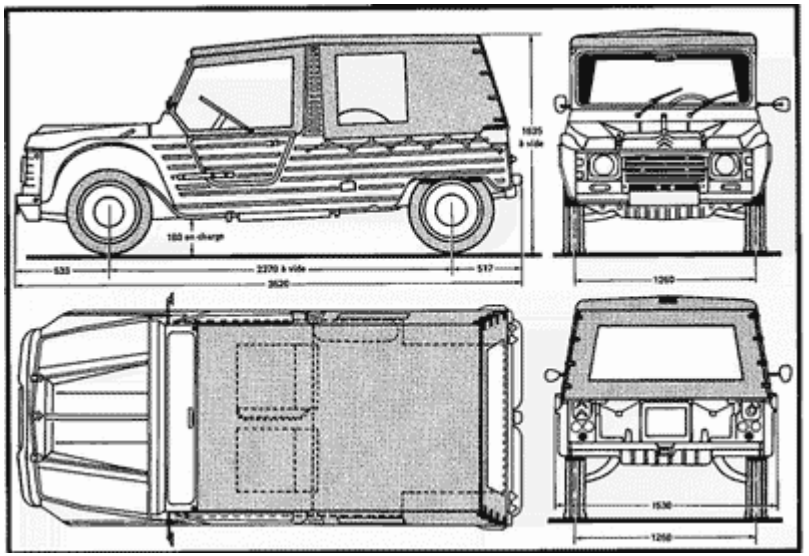
Klasse mit Beispielmethoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Fahrer {

  public String name;
  public int runde;
  public int platz;
  public double rundenzeit;
  public double geschwindigkeit;

  public void beschleunigen(){
    geschwindigkeit = geschwindigkeit + 0.25;
  }

  public void bremsen(){
    geschwindigkeit = geschwindigkeit - 1.4;
  }

  public void starteNeueRunde(){
    rundenzeit = 0.0;
  }
}
```

WIRTSCHAFTS INFORMATIK

Klassen als Baupläne von Objekten



- **Klassen** sind das wichtigste Merkmal objektorientierter Programmiersprachen
- Eine Klasse definiert einen neuen **strukturierten Datentyp**, beschreibt die **Modellierung** der Objekte und gibt somit den für gleichartige Objekte den Bauplan an. Eine Klasse modelliert im Wesentlichen zwei Dinge:
  - **Attribute** (was Objekte dieser Klasse **haben**)
  - **Methoden** (was Objekte dieser Klasse **können**)
- Jedes Objekt ist ein Exemplar einer Klasse und hat eine **Identität**

Eine Klasse, beliebige Exemplare (Objekte)

TECHNISCHE UNIVERSITÄT DARMSTADT

Name: **Diddy Kong**
Runde: **1**
Platz: **1**

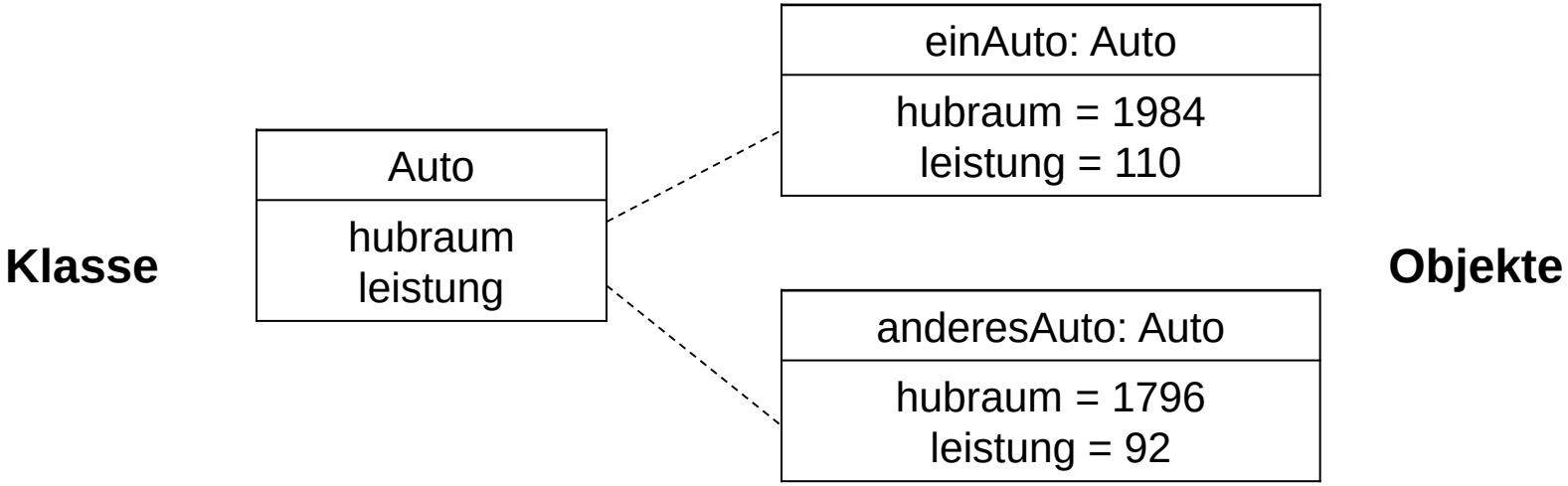
Name: **Toad**
Runde: **1**
Platz: **2**


```
public class MarioKart {
  public static void main(String[] args){
    Fahrer einFahrer = new Fahrer();
    einFahrer.name = "Toad";
    einFahrer.runde = 1;
    einFahrer.platz = 2;

    Fahrer andererFahrer = new Fahrer();
    andererFahrer.name = "Diddy Kong";
    andererFahrer.runde = 1;
    andererFahrer.platz = 1;
  }
}
```


WIRTSCHAFTS INFORMATIK

Klassen vs. Objekte

- Die **Klasse** ist der **Datentyp**, die **Objekte** sind die **Werte**!
- Jedes Objekt ist **Instanz** genau einer Klasse, aber eine Klasse kann beliebig viele Instanzen besitzen
- Alle Objekte einer Klasse besitzen die gleichen **Methoden** und haben daher das gleiche Verhalten. Alle Objekte einer Klasse haben die gleichen **Attribute**, allerdings mit unterschiedlichen Werten (Zustand)



Attribute und Methoden von Objekten

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Attribute von Objekten**
  - Variablen, die zu einem Objekt gehören (z. B. Leistung, Hubraum)
  - Die Attribute speichern Daten, die ein Objekt verwalten kann

- **Methoden von Objekten**
  - Methoden definieren das Verhalten, das ein Objekt aufweisen kann.
  - Methoden können …
    - … den Zustand des Objekts verändern (die Methoden eines Objektes verändern die Werte seiner Attribute)
    - … den Wert eines Attributs des Objekts liefern

In Java eigene Klassen erstellen

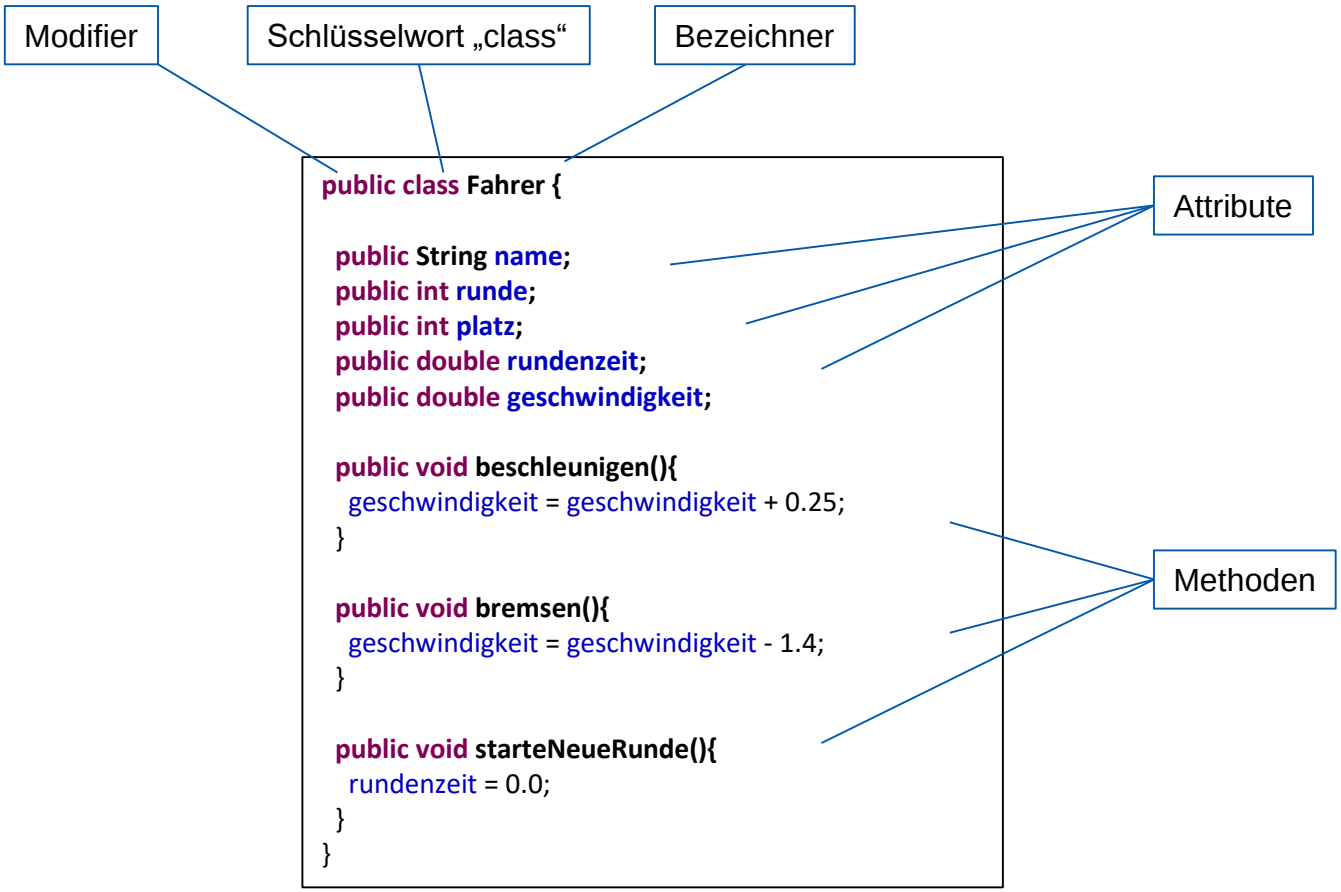
TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Definition einer Klasse in Java besteht aus:
  - einem Modifier
  - dem Schlüsselwort `class`
  - einem Bezeichner
  - Attributen und Methoden

- Syntax:

```
MODIFIER class BEZEICHNER {
  ATTRIBUTE
  METHODEN
}
```

Beispiel: Klassendefinition



Objektvariablen deklarieren

TECHNISCHE UNIVERSITÄT DARMSTADT

- Jede Java-Klasse definiert einen strukturierten Datentyp (Objekttyp), von dem Variablen angelegt werden können.
- Beispiel:

```
Auto einAuto;
```

- Diese Deklaration erzeugt noch kein Objekt der entsprechenden Klasse!
  → Objekte müssen explizit vom Programmierer erzeugt werden

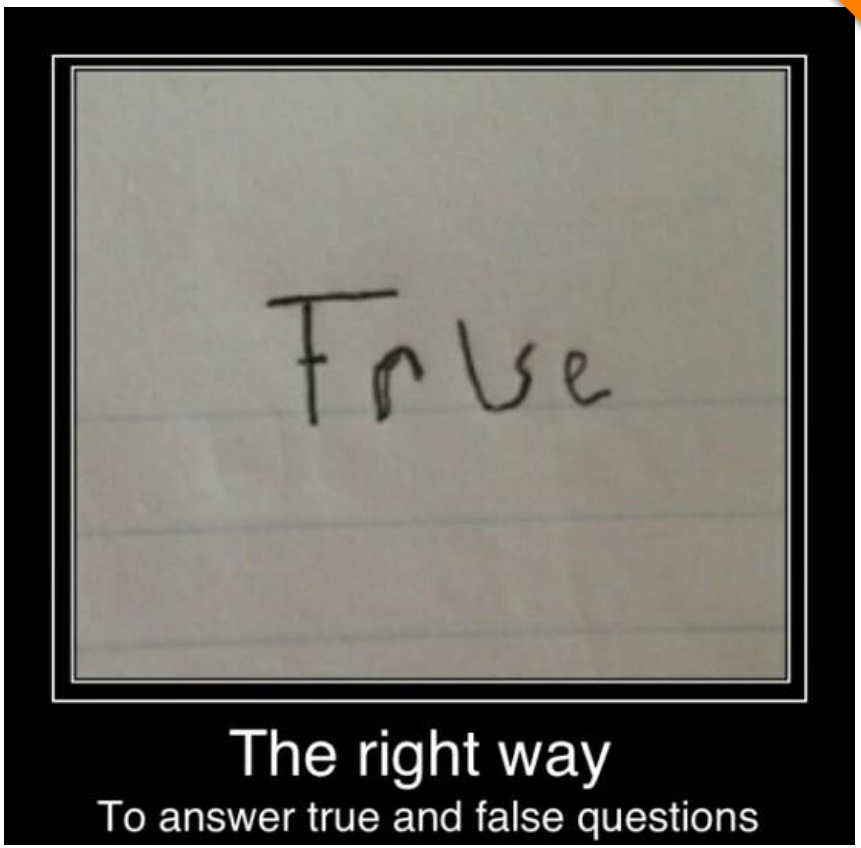
True oder false?

1. Jedes Objekt ist das Exemplar einer Klasse

2. Für ein Attribut einer Klasse muss ein Datentyp festgelegt werden

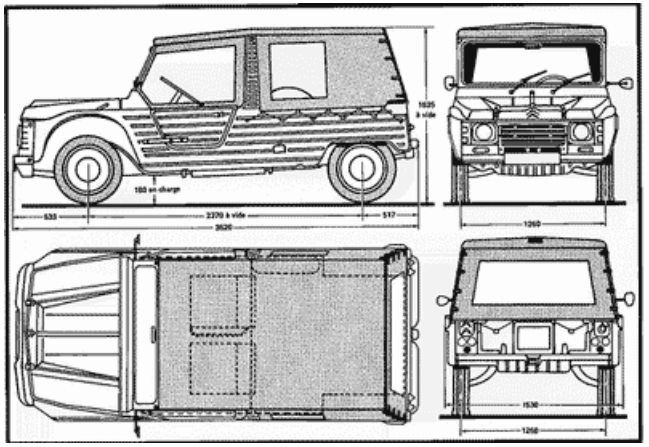
3. Klassen können zur Definition eigener strukturierter Datentypen verwendet werden

4. Für ein Attribut einer Klasse kann auch eine selbst definierte Klasse als Datentyp festgelegt werden



Konstruktoren erzeugen die Objekte einer Klasse

- Konstruktoren sind besondere Methoden und…
  - … können **Parameter** übergeben bekommen.
  - … dienen dazu, Objekte zu **erzeugen** und Attribute zu **initialisieren**.
  - … sind immer **öffentlich**, haben **keinen Rückgabetyp** und **heißen wie ihre Klasse.**
  - … werden mittels **`new`** aufgerufen
- Wird kein eigener Konstruktor definiert, legt Java einen Standard-Konstruktor an.



Beispiel: Konstruktoren

```
public class Auto {
  public int hubraum;
  pbulic int leistung;
  public int garantieEnde;

  public Auto(int h, int l) {
    hubraum = h;
    leistung = l;
  }
  ...
}
```

Konstruktor

Das Attribut hubraum wird mit dem im Parameter h übergebenen Wert belegt.

Aufruf: `Auto meinAuto = new Auto(2200, 110);`

Objekte verwenden

TECHNISCHE UNIVERSITÄT DARMSTADT

Zugriff auf Attribute und Methoden

- Objektmethoden müssen immer mit Bezug zu einem Objekt aufgerufen werden

```
objekt.methode()
```

- Ebenso kann auf Attribute nur mit Bezug zu einem Objekt zugegriffen werden

```
objekt.attribut
```

- Beispiele:

```
einAuto.getModell()
einAuto.modell
```

- **Ausnahme**: Innerhalb der Klasse kann auf die eigenen Attribute/Methoden direkt zugegriffen werden (ohne „`objekt.`“)
- **Achtung**: Bevor auf ein Objekt zugegriffen wird, muss es zunächst erzeugt werden

Beispiel: Verwendung von Objekten

TECHNISCHE UNIVERSITÄT DARMSTADT

Name: **Diddy Kong**
Runde: **1**
Platz: **1**

Name: **Toad**
Runde: **1**
Platz: **2**


```
public class Fahrer {

  public String name;
  public int runde;
  public int platz;
  public double rundenzeit;
  public double geschwindigkeit;

  public void beschleunigen(){
    geschwindigkeit = geschwindigkeit + 0.25;
  }

  public void bremsen(){
    geschwindigkeit = geschwindigkeit - 1.4;
  }

  public void starteNeueRunde(){
    rundenzeit = 0.0;
  }
}
```

```
public class MarioKart {
  public static void main(String[] args){
    Fahrer einFahrer = new Fahrer();
    einFahrer.name = "Toad";
    einFahrer.runde = 1;
    einFahrer.platz = 2;

    Fahrer andererFahrer = new Fahrer();
    andererFahrer.name = "Diddy Kong";
    andererFahrer.runde = 1;
    andererFahrer.platz = 1;
  }
}
```


WIRTSCHAFTS INFORMATIK