

Grundlagen der Programmierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Vorlesungsskript zum Sommersemester 2020

7. Vorlesung (8. Juni 2020)

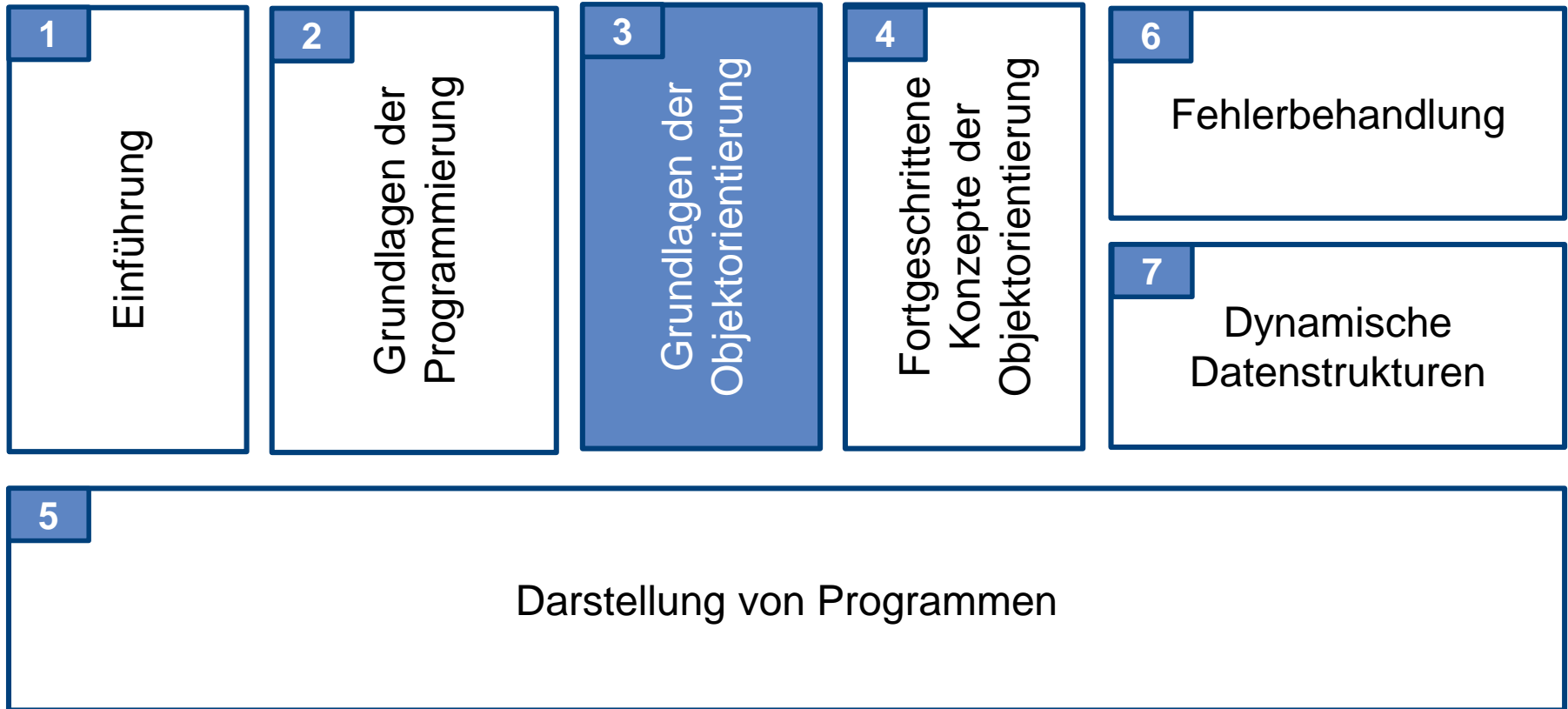


Dr. Jin Gerlach
Christian Olt

Fachgebiet Wirtschaftsinformatik | Software & Digital Business
Fachbereich Rechts- und Wirtschaftswissenschaften
Technische Universität Darmstadt



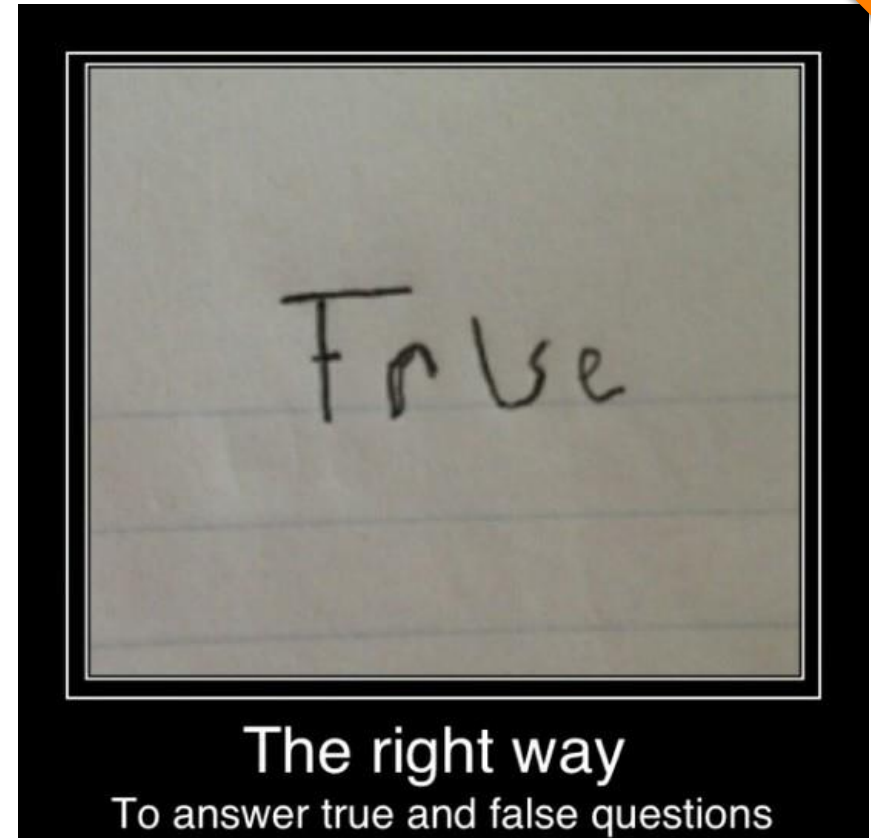
Thematische Übersicht



True oder false?

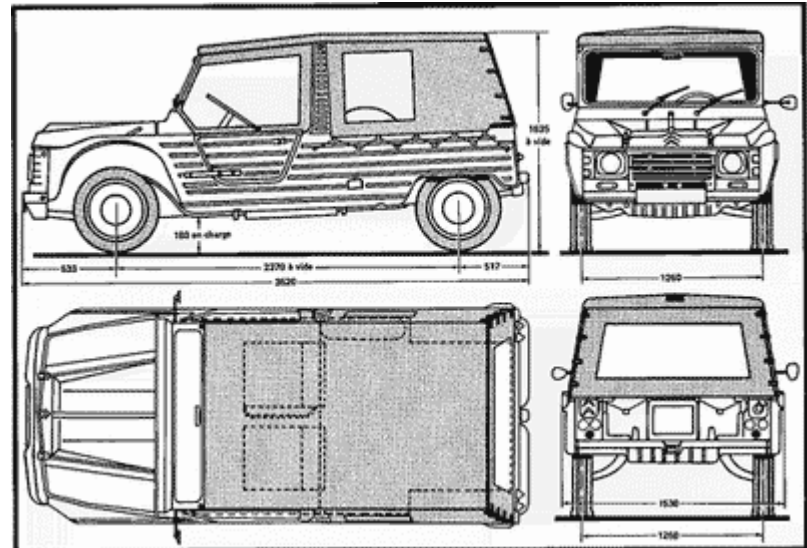


1. Eine Klasse definiert einen strukturierten Datentyp und besteht im Wesentlichen aus Attributen und Methoden.
2. Durch die Definition einer Klasse existiert automatisch auch ein Objekt dieser Klasse.
3. Ein Konstruktor ist eine spezielle Methode einer Klasse, die bei Erzeugung eines Objekts der Klasse aufgerufen wird.
4. Bei der Programmierung einer Klasse muss auch ein Konstruktor programmiert werden.



Klassen als Baupläne von Objekten

- **Klassen** sind das wichtigste Merkmal objektorientierter Programmiersprachen
- Eine Klasse definiert einen neuen **strukturierten Datentyp**, beschreibt die **Eigenschaften** der Objekte und gibt somit den für gleichartige Objekte den Bauplan an. Eine Klasse deklariert 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**





Objektvariablen deklarieren

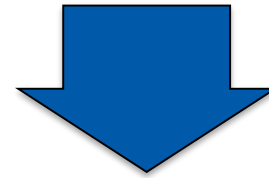
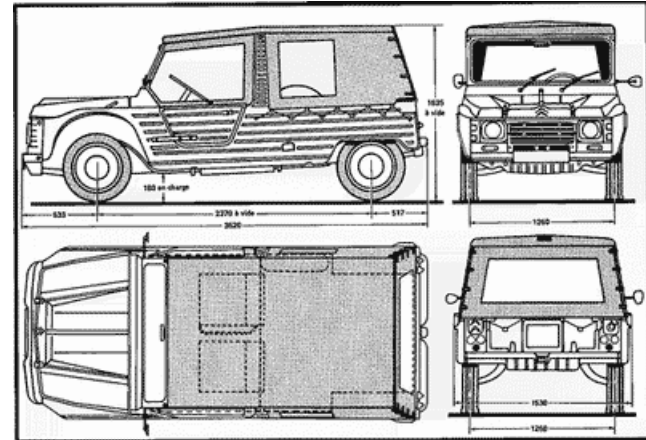
- Jede Java-Klasse definiert einen strukturierten Datentyp (Objekttyp), von dem Variablen angelegt werden können.
- Beispiel:
`Auto einAuto;`
- Diese Deklaration erzeugt noch kein Objekt der entsprechenden Klasse!
→ Objekte müssen explizit vom Programmierer erzeugt werden

```
Auto einAuto = new Auto("Tesla", "Model 3", 460);
```

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 Rückgabotyp** und **heißen wie ihre Klasse**.
 - ... werden mittels **new** aufgerufen
- Wird kein eigener Konstruktor definiert, legt Java einen Standard-Konstruktor an.



Beispiel: Verwendung von Objekten

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;  
    }  
}
```

Kapitel 3: Grundlagen der Objektorientierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Klassenbezogene und objektbezogene Elemente einer Klasse
 - Mathematische Methoden in Java: Die Klasse Math
 - Zeichenketten in Java: Die Klasse String
- Gültigkeitsbereich und Überdeckung von Variablen

Lernziele:

- ☐ Objekt-/Klassenattribute bzw. Objekt-/Klassenmethoden unterscheiden, verstehen und anwenden können
- ☐ Die Java-Klasse Math kennen und verwenden können
- ☐ Die Java-Klasse String kennen und verwenden können
- ☐ Gültigkeitsbereich von Variablen kennen und erklären können.
- ☐ Spezielle Referenz `this` verstehen und anwenden können.

Objektattribute und Klassenattribute

Objektmethoden und Klassenmethoden

- Eine Klasse besteht im Wesentlichen aus zwei Dingen: **Attributen** und **Methoden**
- Es lassen sich zwei Typen von Attributen und Methoden unterscheiden:
 - **Objektattribute** und **Objektmethoden**
(auch: **nicht-statische** Attribute bzw. Methoden)
 - **Klassenattribute** und **Klassenmethoden**
(auch: **statische** Attribute bzw. Methoden)

Beispiel: statische vs. nicht-statische Elemente

```
public class Fahrer {  
  
    public String name;  
    public int runde;  
    public int platz;  
    public double rundenzeit;  
    public double geschwindigkeit;  
  
    public static double beschleunigungsfaktor = 0.25;  
  
    public void beschleunigen(){  
        geschwindigkeit = geschwindigkeit + Fahrer.beschleunigungsfaktor;  
    }  
  
    public void bremsen(){  
        geschwindigkeit = geschwindigkeit - 1.4;  
    }  
  
    public void starteNeueRunde(){  
        rundenzeit = 0.0;  
    }  
}
```




Objektattribute vs. Klassenattribute

- Ein **Objektattribut** gehört zu einem konkreten Objekt, somit:
 - Jedes Objekt verfügt über eine eigene Variable für das Attribut
 - Der Wert des Attributs kann für jedes Objekt verschieden sein
- **Deklaration** durch **Weglassen** des Modifiers `static`:
`public int runde;`
- Zugriff erfolgt über den Namen der Objektvariablen (kann innerhalb der Klasse weggelassen werden):
`Fahrer einFahrer = new Fahrer();`
`int x = einFahrer.runde;`
- Ein **Klassenattribut** gehört zu der jeweiligen Klasse, somit:
 - Alle Objekte der Klasse teilen sich dieselbe Variable für das Attribut (Folge: der Wert des Attributs ist bei allen Objekten gleich)
- **Deklaration** durch **Verwendung** des Modifiers `static`:
`public static double beschleunigungsfaktor = 0.25;`
- Zugriff auf Klassenattribute erfolgt über den Klassennamen (kann innerhalb der Klasse weggelassen werden):
`double x =`
`Fahrer.beschleunigungsfaktor;`

Entscheidung: Wann Klassen-, wann Objektattribute?

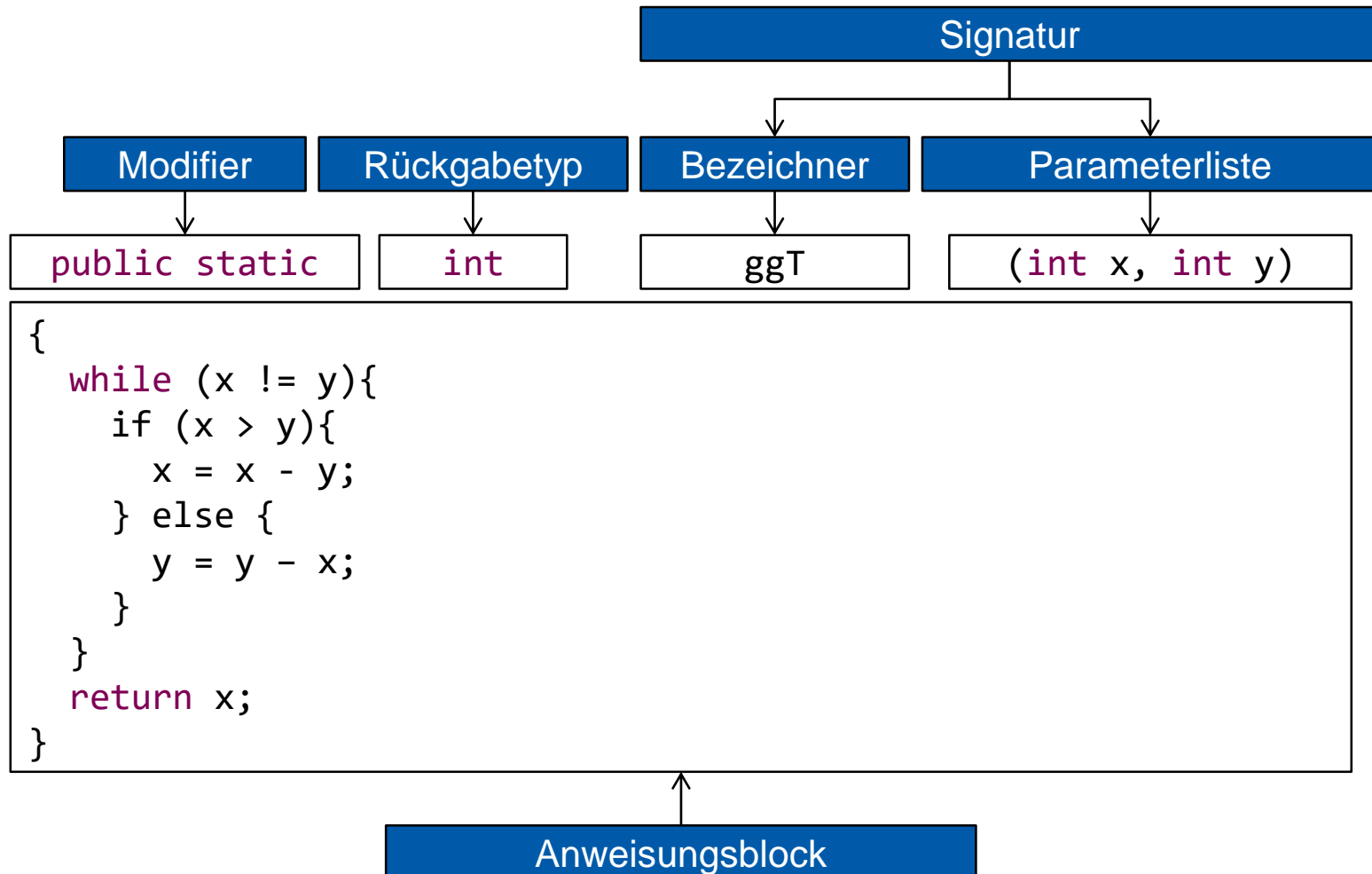
- **Objektattribute**, wenn der Wert objekt-spezifisch ist, z. B.:
 - Fahrer und Name
 - Auto und PS-Anzahl
 - ...
- **Klassenattribute**, wenn der Wert für alle Objekte einer Klasse gleich sein soll, z.B.:
 - Konstanten (bspw. Kreiszahl Pi)
 - Zähler von Instanzen einer Klasse
 - ...



☐ Ja
☐ Nein
☐ Vielleicht



Elemente einer Methode



Objektmethoden vs. Klassenmethoden

- **Objektmethoden** können erst aufgerufen werden, nachdem ein Objekt der jeweiligen Klasse erzeugt wurde
- Objektmethoden werden deklariert, indem der **Modifier** `static` **weggelassen** wird, z. B.:
- **Klassenmethoden** (auch: statische Methoden) werden ohne Bezug zu einem Objekt aufgerufen, z. B.:
`int i = MyMath.ggT(64, 48);`
- Eine Methode wird als statisch deklariert, indem der **Modifier** `static` **verwendet** wird, z. B.:

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

```
public static int ggT (int x, int y)  
{  
    //...  
    return x;  
}
```


Entscheidung: Wann Klassen-, wann Objektmethoden?

- **Objektmethoden**, wenn die Methode mit Objektattributen arbeitet
- **Klassenmethoden**, wenn die Methode nicht auf Objektattribute zugreift, klassische Anwendungsfälle:
 - Methoden, die nur mit den **übergebenen Parametern** arbeiten
(z.B. Methoden in Klasse Math)
 - Methoden, die nur mit Klassenattributen arbeiten



Zahlreiche mathematische Methoden werden durch die Klasse `Math` bereitgestellt, beispielsweise:

```
long p = Math.round(3.14);
```

```
double ceil(double x)
```

ganzzahliges Aufrunden der Zahl x

```
double floor(double x)
```

ganzzahliges Abrunden der Zahl x

```
int max(int x, int y)
```

Maximum von x und y

```
int min(int x, int y)
```

Minimum von x und y

```
double pow(double x, double y)
```

Potenzfunktion x^y

```
double sqrt(double x)
```

Wurzel von x

```
double abs(double x)
```

Absolutbetrag

```
long round(double x)
```

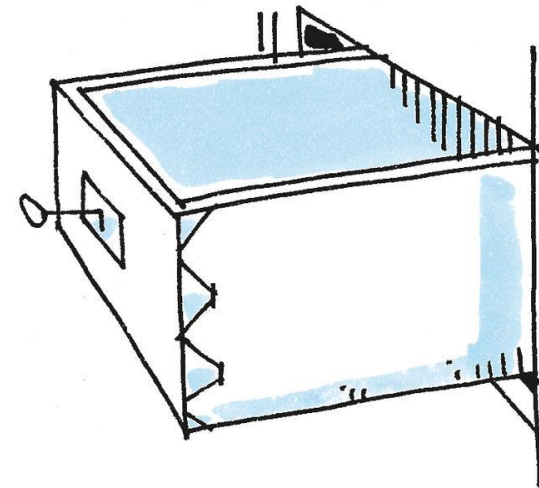
rundet `double` nach `long`

```
...
```



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)





Beispiel: Verwendung von Objekten

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;
    }
}
```

Zeichenketten in Java: Die Klasse String

- Zeichenketten (Strings) sind in Java nicht als einfacher Datentyp, sondern als **Klasse** realisiert
- Um Strings zu erzeugen reicht eine einfache Zuweisung (Dass hier nicht unbedingt ein Konstruktor aufgerufen werden muss ist ein absoluter Sonderfall!)

```
String str = "Hans";
```

entspricht

```
String str = new String("Hans");
```

- Die einzelnen Positionen in einem String sind durchnummeriert:

```
String s = "Max Mustermann";
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
M	a	x		M	u	s	t	e	r	m	a	n	n

Ausgewählte Methoden der Klasse String

Methode	Rück- gabotyp	Beschreibung
length()	int	Liefert die Länge des Strings
charAt(int index)	char	Liefert das Zeichen an der Position <i>index</i> des Strings
equals(String s)	boolean	Gibt true zurück, wenn der übergebene String <i>s</i> und der aktuelle String (auf dem die Methode aufgerufen wird) übereinstimmen
indexOf(String zeichen, int anfang)	int	Liefert die erste Position von <i>zeichen</i> innerhalb des Strings ab der Stelle <i>anfang</i> bzw. -1, falls <i>zeichen</i> nicht vorkommt
replace(char z1, char z2)	String	Ersetzt alle <i>z1</i> durch <i>z2</i>
toUpperCase()	String	Gibt den String in Großbuchstaben zurück
substring(int pos, int ende)	String	Liefert den Stringinhalt ab der Stelle <i>pos</i> (inklusive) bis zur Stelle <i>ende</i> (exklusive)

Beispiel: Methoden der Klasse String

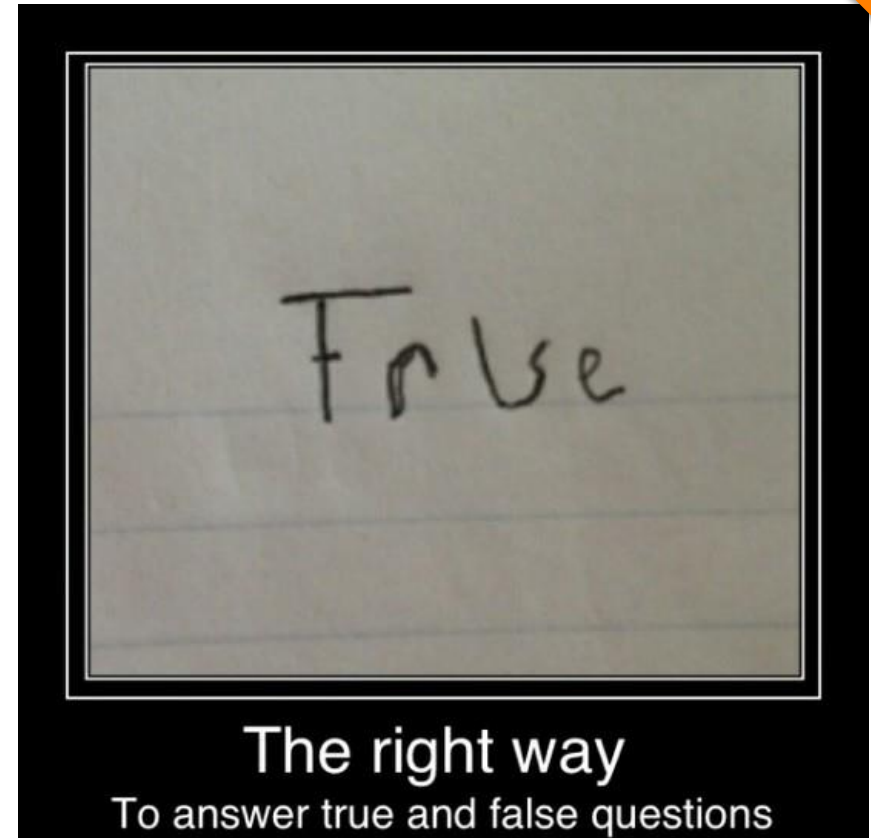


```
public class StringMethods {  
    public static void main(String[] args) {  
        String s = "Haus";  
  
        int i = s.length();           // i = 4  
        char c = s.charAt(2);         // c = 'u'  
        boolean b = s.equals("aus"); // b = false  
        int x = s.indexOf("abc", 0);  // x = -1  
        s = s.toUpperCase();           // s = "HAUS"  
        s = s.replace('H', 'M');       // s = "MAUS"  
        s = s.substring(1, 4);         // s = "AUS"  
    }  
}
```

True oder false?



1. Das Schlüsselwort „static“ kennzeichnet Klassenmethoden und Klassenattribute.
2. Vor dem Aufrufen einer Klassenmethode muss kein Objekt der Klasse erzeugt werden.
3. Mathematische Konstanten sollten als Objektattribute gespeichert werden.
4. Die Klasse String besitzt verschiedene Objektmethoden, die auf Zeichenketten in Java aufgerufen werden können.



- Jede Variable hat innerhalb eines Programms einen bestimmten **Gültigkeitsbereich**, in dem sie **existiert** (d.h. ihr Wert ist im Speicher des Computers abgelegt). Wird der Gültigkeitsbereich (z.B. eine Methode) verlassen, wird die Variable **aus dem Speicher entfernt**.
- Je nach Typ haben Variablen unterschiedliche Gültigkeitsbereiche. Man unterscheidet drei Typen von Variablen:
 - **Klassenvariablen** (Klassenattribute) sind im ganzen Programm gültig und existieren auch ohne dass eine Instanz der Klasse angelegt wurde.
 - **Objektvariablen** (Objektattribute) existieren in Verbindung mit einem vorher instanziierten Objekt. Innerhalb der Klassendefinition existieren sie nur für Objektmethoden.
 - **Lokale Variablen** sind nur in dem Block (und allen untergeordneten Blöcken) gültig, in dem sie deklariert wurden, z.B. in einer Methode oder einer Schleife.



Gruppieren von Anweisungen mit Blöcken

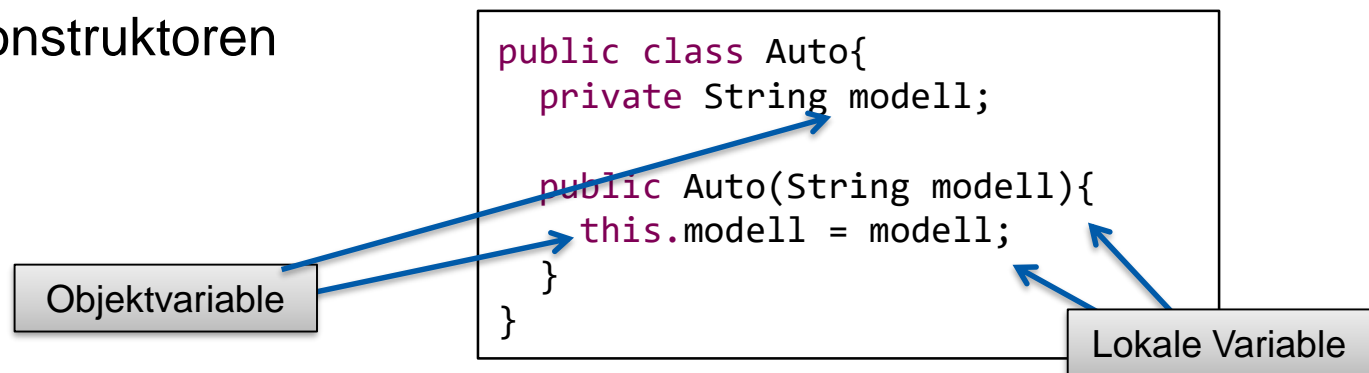
Ein **Block** fasst eine Gruppe von Anweisungen zusammen, die hintereinander ausgeführt werden. Anders gesagt: Ein Block ist eine Anweisung, die in geschweiften Klammern { /* ... */ } eine Folge von Anweisungen zu einer neuen Anweisung zusammenfasst:

```
{  
Anweisung1;  
Anweisung2;  
...  
}
```

Hinweis: Damit ist das Verschachteln von Kontrollstrukturen (und Blockanweisungen) möglich!

Überdeckung durch lokale Variablen

- Klassen- und Objektvariablen können durch lokale Variablen überdeckt werden, wenn eine lokale Variable mit dem selben Namen deklariert wird.
- Mit dem Klassennamen kann auf eine überdeckte Klassenvariable zugegriffen werden.
- Mit der Selbstreferenz **this** kann auf eine überdeckte Objektvariable zugegriffen werden
- Tritt häufig in Konstruktoren auf:



- In jeder Objektmethode und jedem Konstruktor steht eine spezielle Referenz mit dem Namen `this` bereit, die auf das eigene Objekt zeigt
- Wird verwendet:
 - Um auf Objektattribute zuzugreifen, die durch gleichnamige Parameter bzw. lokale Variablen verdeckt werden
 - Mit der `this`-Referenz lässt sich eine Referenz auf das eigene Objekt zurückgeben

Kapitel 3: Grundlagen der Objektorientierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

- Vererbung in der Programmiersprache Java
- Überschreiben von Methoden in einer Vererbungshierarchie
- Polymorphismus: Statische und dynamische Datentypen

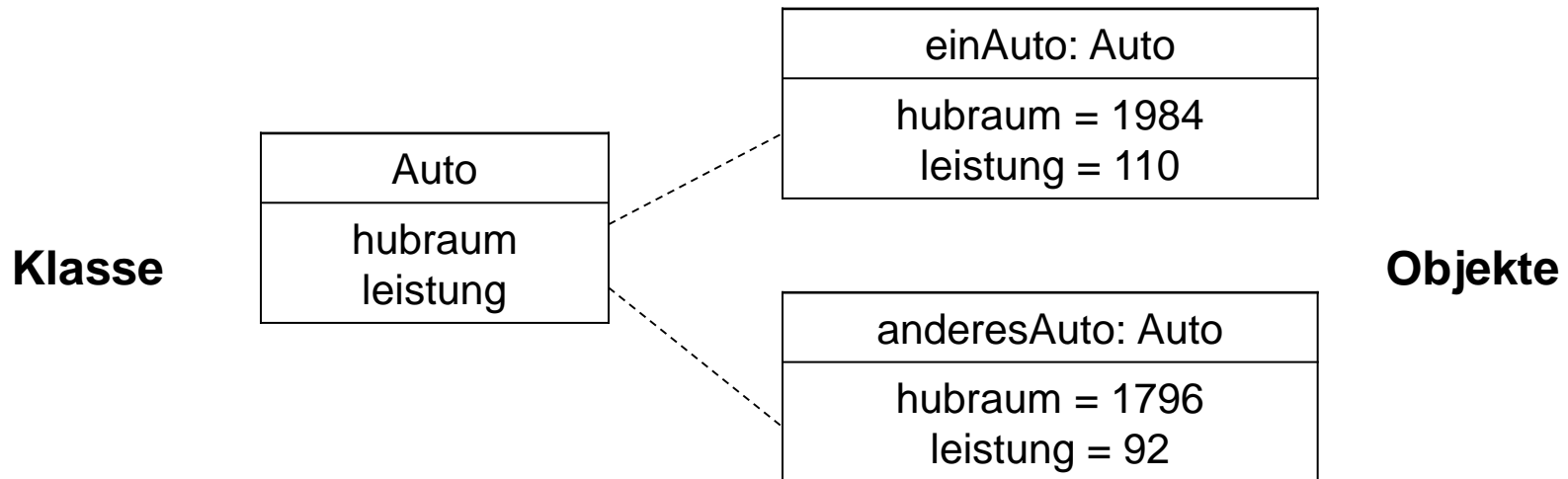
Lernziele:

- Vererbung verstehen und erklären können
- Erbende Klassen in Java erstellen und nutzen können
- Das Überschreiben von Konstruktoren und Methoden in Vererbungshierarchien erklären und sinnvoll anwenden können.
- Statische und dynamische Datentypen unterscheiden können.



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)



Beispiel: Autos, LKWs und Fahrzeuge

- **Ziel: Programm zur Verwaltung des Fuhrparks der TU Darmstadt**
- Es sollen Autos und LKWs verwaltet werden
- Das Fuhrparkmanagement möchte zu den **LKWs** neben der Modellbezeichnung, der Leistung und dem Hubraum auch die maximale Zuladung speichern
- Für die **Autos** sollen neben der Modellbezeichnung, der Leistung und dem Hubraum auch die Anzahl der Sitzplätze und Türen gespeichert werden

Beispiel: Autos und LKWs

Lösung ohne Vererbung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
public class Auto {  
    public String modell;  
    public int ps;  
    public int sitzplaetze;  
    public int tueren;  
  
    public Auto(String m) {  
        modell = m;  
    }  
}
```

```
public class LKW{  
    public String modell;  
    public int ps;  
    public int zuladung;  
  
    public LKW(String m) {  
        modell = m;  
    }  
}
```

Autos und LKWs als zwei Klassen?

Argumente für EINE Klasse	Argumente für ZWEI Klassen
Es gibt Attribute und Methoden, die sowohl Autos als auch LKWs besitzen. Diese wurden doppelt implementiert	Es gibt Attribute und Methoden von LKWs, die es bei Autos nicht gibt und es gibt Attribute und Methoden von Autos, die es bei LKWs nicht gibt

- **Lösung: Vererbung** (bildet eine „*ist-eine-Art-von-Beziehung*“ ab)
 - Ein Auto ist ein **Fahrzeug**
 - Ein LKW ist ein **Fahrzeug**
- Gleiche Attribute und Methoden werden in die übergeordnete Klasse „Fahrzeug“ verschoben und sowohl von Autos als auch von LKWs **geerbt**

Vererbung in der Objektorientierung

- Das Konzept der **Vererbung** erlaubt es auf der „Klassen-Ebene“ nicht nur einzelne Klassen zu definieren, sondern auch **Beziehungen** zwischen verschiedenen Klassen zu modellieren.
- Vererbung bildet eine „ist-eine-Art-von-Beziehung“ ab:
 - Apfel **ist eine Art von** Obst
 - Birne **ist eine Art von** Obst
 - Auto **ist eine Art von** Fahrzeug
 - LKW **ist eine Art von** Fahrzeug

Eine **generalisierende Ober-Klasse** abstrahiert von **spezialisierenden Sub-Klassen**, indem sie Gemeinsamkeiten dieser Klassen zusammenfasst

