Grundlagen der Programmierung

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorlesungsskript zum Sommersemester 2020

8. Vorlesung (15. Juni 2020)

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

WIRTSCHAFTS INFORMATIK

Kapitel 3: Grundlagen der Objektorientierung

TECHNISCHE UNIVERSITÄT DARMSTADT

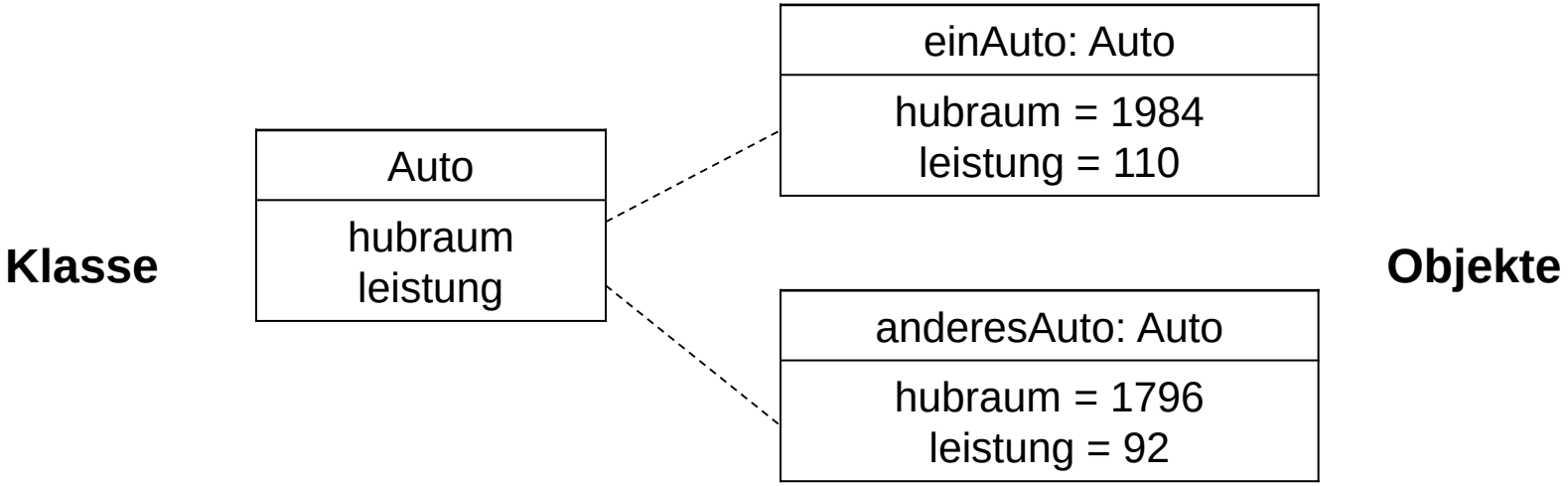
- Vererbung in der Programmiersprache Java
- Überschreiben von Methoden in einer Vererbungshierarchie
- Polymorphismus: Statische und dynamische Datentypen

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

TECHNISCHE UNIVERSITÄT DARMSTADT

- **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 und der Leistung auch die maximale Zuladung speichern
- Für die **Autos** sollen neben der Modellbezeichnung und der Leistung 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?

TECHNISCHE UNIVERSITÄT DARMSTADT

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

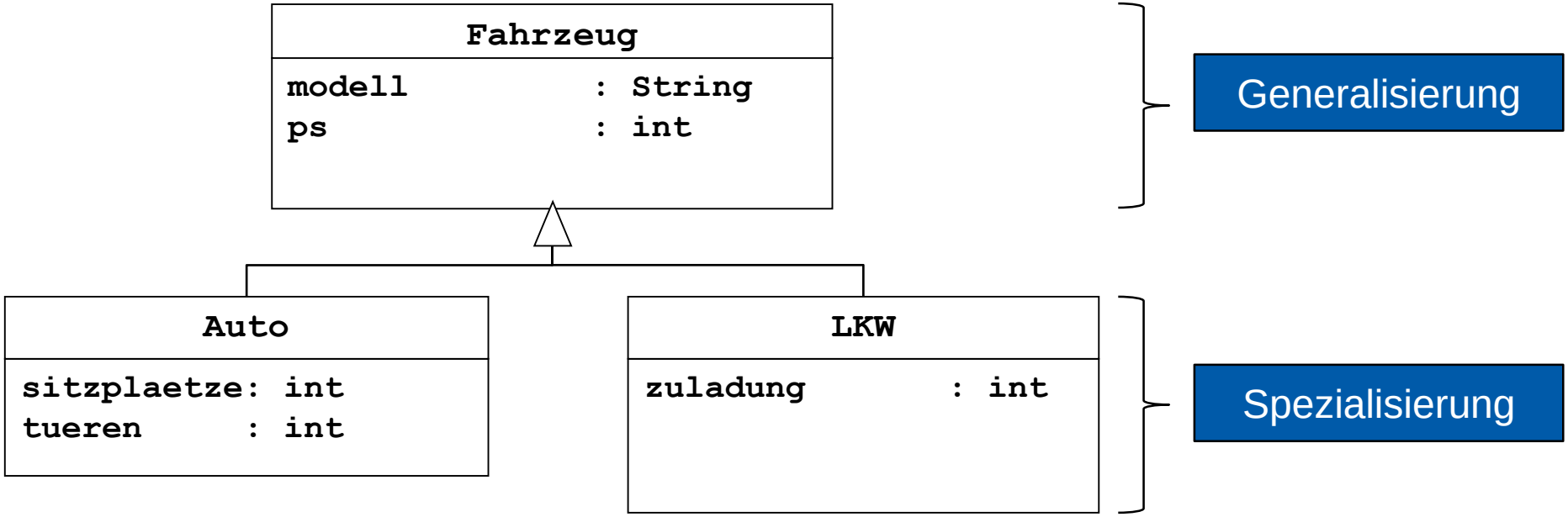
TECHNISCHE UNIVERSITÄT DARMSTADT

- 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

Generalisierung

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

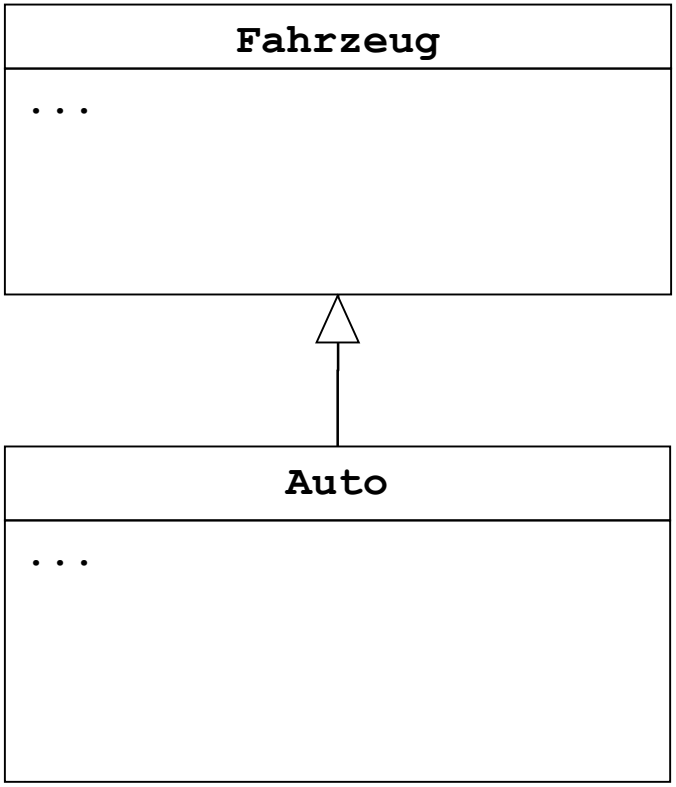


Eigenschaften des Programmierkonzepts „Vererbung“

TECHNISCHE UNIVERSITÄT DARMSTADT

- Sub-Klassen **erben** Attribute und Methoden der Ober-Klasse
  - Jede **Methode** der Ober-Klasse ist automatisch auch eine Methode der abgeleiteten Klasse (sie hat dieselbe Signatur und dieselbe Implementierung)
  - Jedes **Attribut** der Ober-Klasse ist auch ein Attribut der abgeleiteten Klasse
  - Achtung: **Konstruktoren** werden nicht vererbt
- **Vorteile dieses Programmierkonzepts**
  - Wiederverwendung
  - Schrittweise Entwicklung vom Generellen zum Speziellen

Vererbung in Java



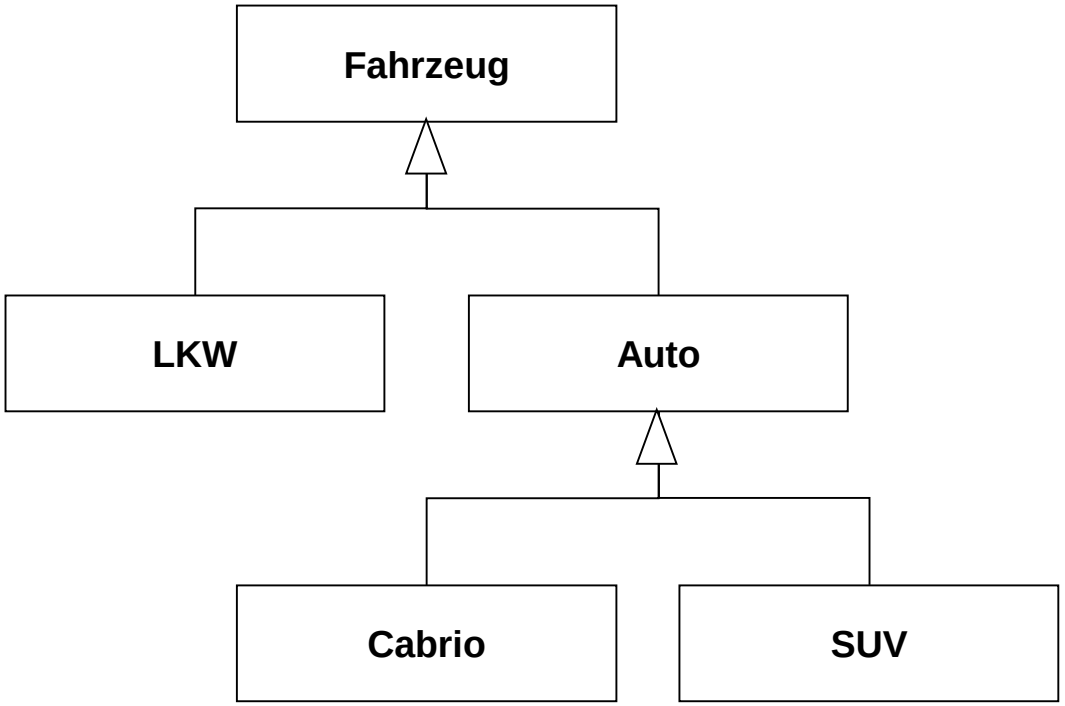

```
public class Fahrzeug{

   ...

}
```

```
public class Auto extends Fahrzeug{

   ...

}
```

Transitive Vererbung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Vererbung ist **transitiv**: Von **Sub-Klassen** können wiederum neue **Sub-Sub-Klassen** abgeleitet werden
- Diese **Sub-Sub-Klassen** erben auch die Attribute und Methoden, welche die eigene **Ober-Klasse** von ihrer **Ober-Klasse** geerbt hat



WIRTSCHAFTS INFORMATIK

Vererbung in Java

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Syntax:**

```
public class Sub-Klasse extends Ober-Klasse {...}
```

- **Sub-Klasse erbt:**
  - Alle **Attribute**
  - Alle **Methoden**
  - **Keinen Konstruktor**
- **Einschränkungen**:
  - Ableitung nur von **einer** einzigen Klasse möglich

Vererbung und Konstruktoren

- Konstruktoren werden **nicht** vererbt! Daher müssen Sub-Klassen **Konstruktoren neu implementieren**
- Jeder Konstruktor einer Sub-Klasse muss genau einen Konstruktor der Ober-Klasse aufrufen
- In der Sub-Klasse heißt der Konstruktor der Ober-Klasse `super(Parameterliste)` und verfügt über die Parameter des jeweiligen Konstruktors der Ober-Klasse
- Der `super`-Aufruf muss die erste Anweisung im Konstruktor der Sub-Klasse sein

Beispiel: Autos, LKWs und Fahrzeuge

Lösung mit Vererbung

TECHNISCHE UNIVERSITÄT DARMSTADT

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

„Ist eine Art von“ „Ist eine Art von“

```
public class Auto extends Fahrzeug{
  public int sitzplaetze;
  public int tueren;
  public Auto(String m){
    super(m);
  }
}
```

```
public class LKW extends Fahrzeug{
  public int zuladung;
  public LKW(String m){
    super(m);
  }
}
```


WIRTSCHAFTS INFORMATIK

Überschreiben von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Sub-Klassen können geerbte Methoden neu implementieren**
- Dieser Vorgang wird als **Überschreiben** bezeichnet
  - Hierzu wird in der Sub-Klasse eine Methode mit derselben Signatur implementiert, sie überschreibt die entsprechende Methode aus der Ober-Klasse
  - Beim Aufruf einer überschriebenen Methode auf (Objekten) der Sub-Klasse wird die neue Implementierung der überschreibenden Methode aufgerufen
  - Die Sub-Klasse kann auf die überschriebenen Methoden der Ober-Klasse über die Referenz `super` zugreifen:

```
super.ueberschriebeneMethode(...);
```

Beispiel: Überschreiben von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Fahrzeug{

  public void druckeInfo() {
    System.out.println(modell);
    System.out.println(ps);
  }

  ...
}
```

```
public class Auto extends Fahrzeug {

  public void druckeInfo() {
    super.druckeInfo();
    System.out.println(tueren);
    System.out.println(sitzplaetze);
  }
  ...
}
```

```
public class MyClass {
  public static void main(String[] args) {
    Auto auto = new Auto("VW Golf", 50, 5, 5);
    auto.druckeInfo();
    Fahrzeug truck = new Fahrzeug("MAN TGS", 300);
    truck.druckeInfo();
} }
```

Beispiel: Überschreiben von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Fahrzeug{

  public void druckeInfo() {
    System.out.println(modell);
    System.out.println(ps);
  }

  ...
}
```

```
public class Auto extends Fahrzeug {

  public void druckeInfo() {
    super.druckeInfo();
    System.out.println(tueren);
    System.out.println(sitzplaetze);
  }
  ...
}
```

```
public class MyClass {
  public static void main(String[] args) {
    Auto auto = new Auto("VW Golf", 50, 5, 5);
    auto.druckeInfo();
    Fahrzeug truck = new Fahrzeug("MAN TGS", 300);
    truck.druckeInfo();
} }
```


WIRTSCHAFTS INFORMATIK

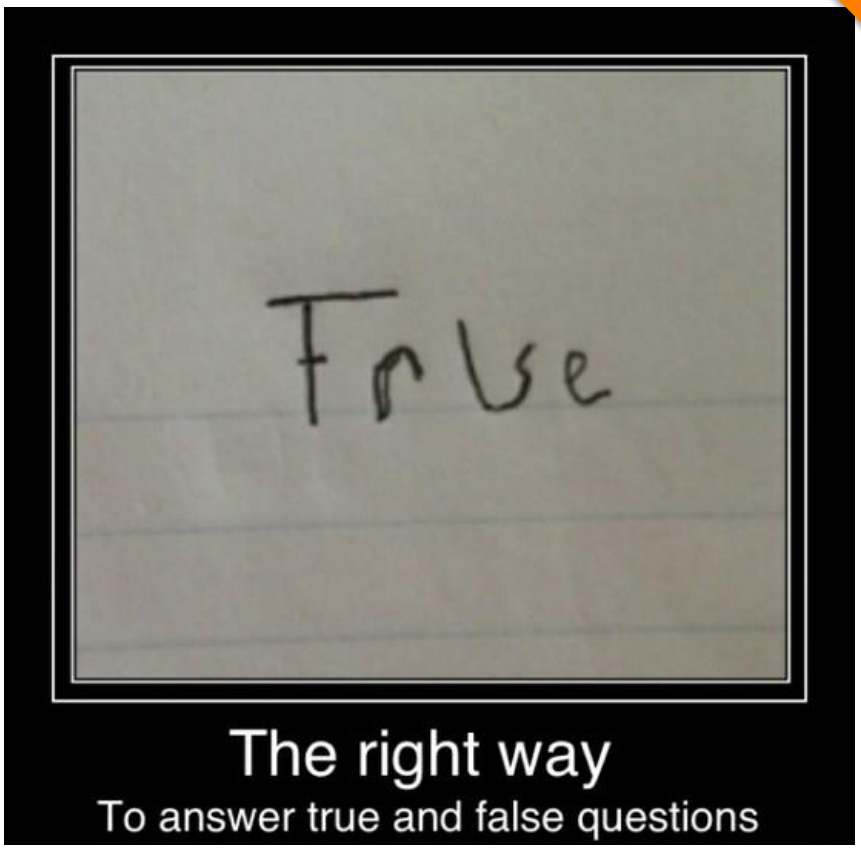
True oder false?

1. Eine Sub-Klasse kann von beliebig vielen Ober-Klassen direkt erben

2. Beim Vererben werden die Attribute aber nicht die Methoden der Ober-Klasse übernommen

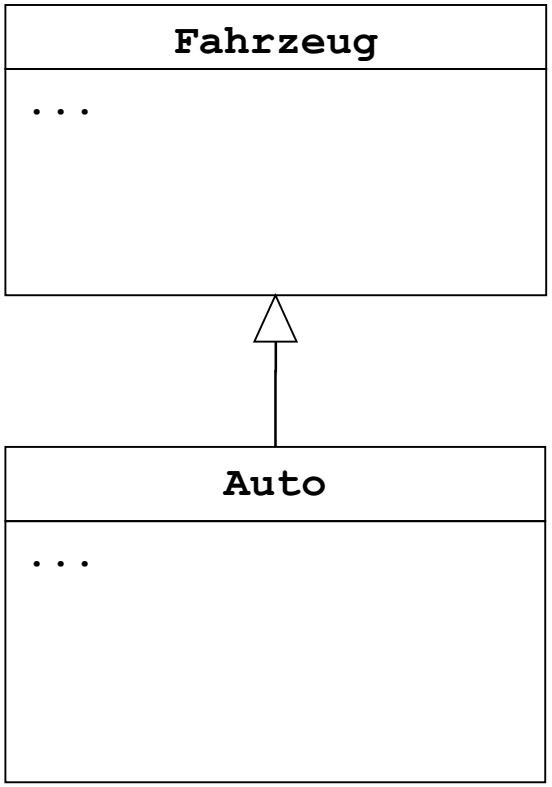
3. Eine überschreibende Methode muss immer die überschriebene Methode der Ober-Klasse aufrufen

4. Wird eine Methode der Ober-Klasse von einer Sub-Klasse nicht überschrieben, so besitzt die Sub-Klasse diese Methode nicht



Polymorphismus aka „Dynamisches Binden“

- **Sub-Klasse ist eine Spezialisierung der Ober-Klasse**
- **Polymorphismus** (griechisch für Vielgestaltigkeit) bedeutet: Eine Variable vom **Datentyp einer Ober**-Klasse kann Objekte
  - vom **eigenen** Datentyp (z. B. Fahrzeug)
  - von **allen Datentypen der Sub-Klassen** (z. B. Auto, LKW) speichern
- **Beispiel:** `Fahrzeug f = new Auto(...);`
- Objekte der Sub-Klasse haben alle Methoden und Attribute der Ober-Klasse
  → Alles, was man mit der Ober-Klasse machen kann, geht auch mit der Sub-Klasse

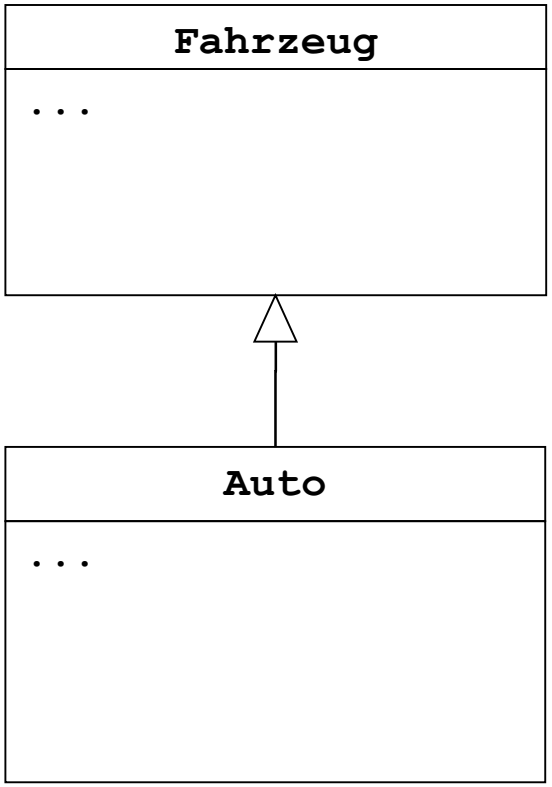


Polymorphismus: Geht nur in eine Richtung!

TECHNISCHE UNIVERSITÄT DARMSTADT

Warum funktioniert folgende Zuweisung nicht?

```
Auto a = new Fahrzeug(„Golf“);
```



Objekte der Sub-Klassen haben alle Methoden und Attribute der Ober-Klasse

Aber: Die **Umkehrung gilt nicht**!
Variablen vom Datentyp der Sub-Klasse können keine Werte vom Datentyp der Ober-Klasse annehmen

WIRTSCHAFTS INFORMATIK

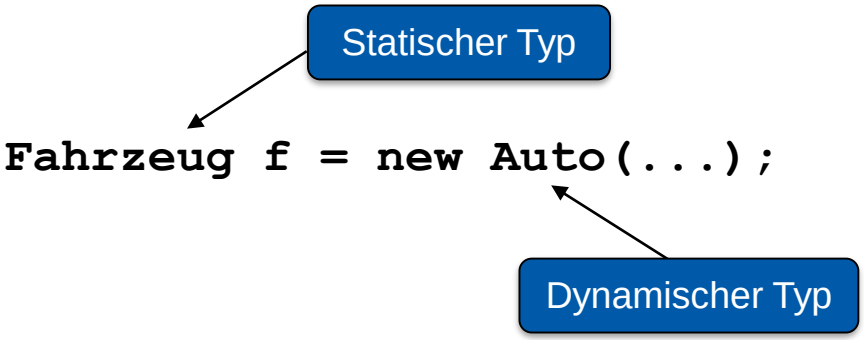
Statische vs. dynamische Datentypen

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Konsequenz aus dem Polymorphismus:**
  Das Objekt, das in einer Variable gespeichert wurde, hat nicht immer den Datentyp, der bei der Deklaration der Variablen angegeben wurde

- **Unterscheidung:**
  - **Statischer Typ** = Datentyp der Variablen
  - **Dynamischer Typ** = Datentyp des Objekts

- Beispiel:



WIRTSCHAFTS INFORMATIK

Polymorphismus und Methodenaufrufe

TECHNISCHE UNIVERSITÄT DARMSTADT

- Methodenaufruf bei polymorphen Variablen
  - Der **statische Typ** bestimmt, **welche Methoden** aufgerufen werden können
  - Der **dynamische Typ** bestimmt, **welche Implementierung** der Methode aufgerufen wird
- Die Auswahl der Methode geschieht zur Laufzeit (dynamisch) und unabhängig von der (statischen) Deklaration
  - Falls die Sub-Klasse die **Methode überschreibt**, wird die überschreibende Methode (der Sub-Klasse) aufgerufen
  - Falls die Sub-Klasse die **Methode nicht überschriebt**, wird die geerbte Methode (der Ober-Klasse) aufgerufen

Beispiel: Verschiedene Items mit unterschiedlichen Funktionen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Fahrer {

  public Item item;
  //…
  public void itemAktivieren(){
    item.aktivieren()
   }
  //…
}
```

```
public class Pilz extends Item {

//…
  public void aktivieren(){
     //Spezifischer Pilzcode
   }
  //…
}
```

```
public class Bananenschale extends Item {

//…
  public void aktivieren(){
     //Spezifischer Bananenschalencode
   }
  //…
}
```

WIRTSCHAFTS INFORMATIK

Beispiel: Polymorphismus

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyClass {
    public static void main(String[] args) {
        // Array anlegen
        Fahrzeug[] f = new Fahrzeug[2];

        f[0] = new Auto("VW Golf", 50, 5, 5);

        f[1] = new LKW("MAN", 250, 5000);

        for (int i=0; i < f.length; i++) {
            f[i].druckeInfo();
            System.out.println();
        }
    }
}
```

Eine Variable vom Typ Fahrzeug kann Auto-Objekte und LKW-Objekte speichern.

Aufgabe: Polymorphismus

Welche Bildschirmausgaben erhalten Sie bei Ausführung der main-Methode?

```
class Oberklasse{
  public void meineMethode() {
    System.out.print(„OberKl");
  }
}
```

```
class SubklasseA extends Oberklasse{
  public void meineMethode() {
    System.out.print("SubA");
  }
}
```

```
class SubklasseB extends Oberklasse{
}
```

```
public class PolyTest {
  public static void main
                    (String[] args) {

    Oberklasse o = new Oberklasse();
    o.meineMethode();    // a

    SubklasseA a = new SubklasseA();
    a.meineMethode();    // b

    o = new SubklasseA();
    o.meineMethode();    // c

    o = new SubklasseB();
    o.meineMethode();    // d
  }
}
```

Explizites Casting

- Ein Wert oder eine Variable eines übergeordneten Datentyps kann **explizit** einer Variable eines untergeordneten Datentyps (mit möglichem Verlust von Informationen) zugewiesen werden
  Beispiel: Bei der Umwandlung eines Gleitkommawertes in einen ganzzahligen Wert, werden die Nachkommastellen einfach weggelassen
- Der gewünschte Typ für eine Typanpassung wird vor der umzuwandelnden Variable in Klammern () angegeben (Casting hat **hohe Priorität**!)

Beispiel:

```
double d = 3.1415;
int n = (int) d;      // n = 3
```

Polymorphismus und Casting

Erweitert

TECHNISCHE UNIVERSITÄT DARMSTADT

- Manchmal wird eine Möglichkeit benötigt, Objekte vom Typ „Sub-Klasse“, die in einer Variablen vom Typ „Ober-Klasse“ gespeichert wurden, in einer Variable vom Typ „Sub-Klasse“ zu speichern.
- **Lösung:**
  - Der Wert der Variablen muss explizit umgewandelt werden (type cast):

```
Fahrzeug f = new Auto("Golf");
Auto a = (Auto) f;
```

  - Falls `f` jedoch kein „Auto“-Objekt enthält, tritt ein Fehler auf
  - Mit dem **Schlüsselwort `instanceof`** kann der dynamische Typ einer Variablen geprüft werden. Es kann also auch geprüft werden, ob ein Type Cast zulässig ist
  - **Beispiel:**

```
if (f instanceof Auto)
    a = (Auto) f;
```

Die Urklasse: `java.lang.Object`

Erweitert

TECHNISCHE UNIVERSITÄT DARMSTADT

- Jede Klasse, die nicht explizit von einer Oberklasse abgeleitet wird, wird in Java automatisch von `java.lang.Object` abgeleitet. Damit ist `java.lang.Object` die **Urklasse**, von der alle anderen Klassen abgeleitet sind!
  - Eine Variable vom Typ `Object` kann Objekte einer beliebigen Klasse aufnehmen!
  - java.lang.Object definiert einige Methoden, die von allen Klassen geerbt werden – Beispiele sind:
    - **`boolean equals(Object o)`** vergleicht zwei Objekte
    - **`String toString()`** gibt eine Repräsentation des Objekts als String zurück *(nicht aus! - z.* B. für System.out.println(...))
    - **`Object clone()`** liefert eine Kopie des Objekts

WIRTSCHAFTS INFORMATIK

Transitive Vererbung

- Vererbung ist **transitiv**: Von **Sub-Klassen** können wiederum neue **Sub-Sub-Klassen** abgeleitet werden
- Diese **Sub-Sub-Klassen** erben auch die Attribute und Methoden, welche die eigene **Ober-Klasse** von ihrer **Ober-Klasse** geerbt hat