Grundlagen der Programmierung

TECHNISCHE UNIVERSITÄT DARMSTADT

Vorlesungsskript zum Sommersemester 2020
10. Vorlesung (29. Juni 2020)

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

WIRTSCHAFTS INFORMATIK

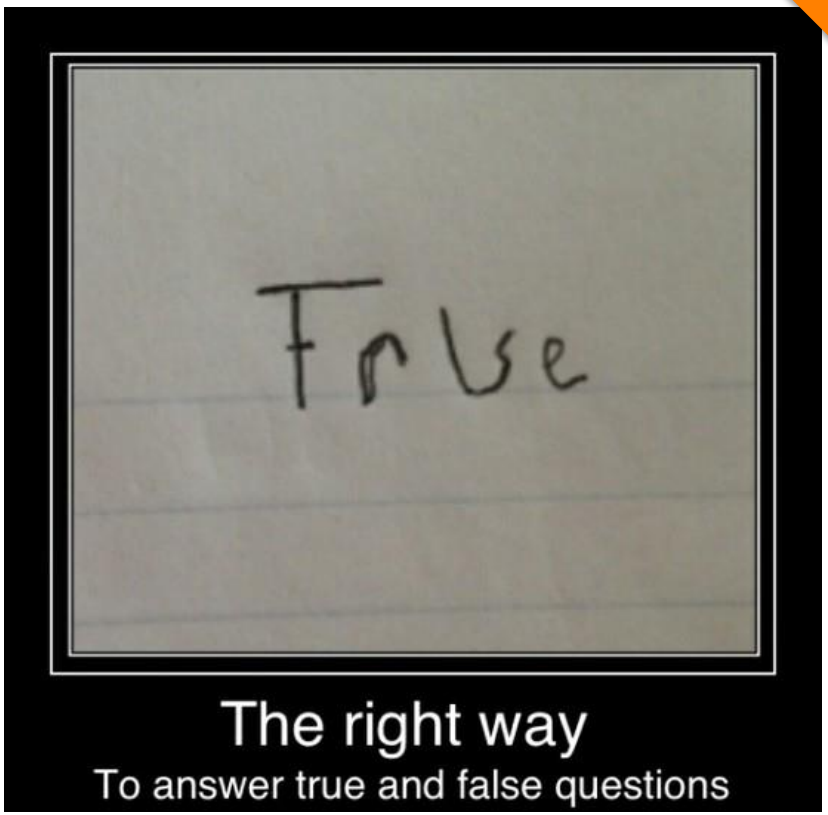
True oder false?

1. Um aus einer Klasse eines Pakets heraus auf Klassen aus anderen Paketen zuzugreifen, müssen diese in die eigene Klasse importiert werden.

2. Auf private Attribute und Methoden kann nur innerhalb der eigenen Klasse zugegriffen werden.

3. Der Vergleichsoperator == vergleicht zwei Objekte hinsichtlich ihrer Speicheradresse.

4. Die von „Object“ geerbte equals-Methode vergleicht zwei als Parameter übergebene Objekte miteinander.



Zugriff auf Pakete

- Direkter Zugriff (über den Klassennamen) nur auf Klassen aus dem **eigenen Paket** und auf Klassen aus Paket `java.lang`
- Es gibt zwei Möglichkeiten zum Zugriff auf Klassen aus anderen Paketen
  - **Möglichkeit 1: Über den vollqualifizierten Namen:**
    ```
    de.tudarmstadt.MyClass c = new de.tudarmstadt.MyClass();
    ```
  - **Möglichkeit 2: Importieren**
    - Import nur einer Klasse :
      ```
      import java.util.Scanner;
      ```
    - Import aller Klassen in Paket `java.util`:
      ```
      import java.util.*;
      ```
    - Die Import-Anweisung steht hinter der `package`-Anweisung und vor dem Klassenkopf!

Modifier: Öffentliche und private Methoden und Attribute

Java erlaubt verschiedene Einschränkungen bzgl. des Zugriffs auf die Variablen / Methoden eines Objekts.

„Öffentlich“ ⟷ „Privat“

`public`: auf öffentliche Variablen / Methoden darf von allen Klassen aus zugegriffen werden

`protected`: auf geschützte Variablen/Methoden darf nur von der eigenen Klasse, von Klassen im eigenen Paket sowie von Objekten der Unterklasse zugegriffen werden

Keine Sichtbarkeits-angabe (**`package`**): auf Variablen / Methoden darf nur im eigenen Paket zugegriffen werden

`private:` auf private Variablen / Methoden darf nur innerhalb der eigenen Klasse zugegriffen werden

Gleichheit von Objekten

- Der Vergleichsoperator `==` überprüft die Übereinstimmung zweier Werte, d.h. bei Objektvariablen werden die gespeicherten Referenzen auf Gleichheit überprüft, aber nicht die Attribute der referenzierten Objekte!
- Beispiel:

```
Auto a = new Auto();
a.modell="Golf 3";
Auto b = new Auto();
b.modell="Golf 3";
System.out.println(a == b); // Ausgabe: false
a = b;
System.out.println(a == b); // Ausgabe: true
```

Vergleich von Objekten mit `equals(...)`

- Durch Überschreiben der Methode `equals(Object o)` kann ein Programmierer festlegen, wann Objekte einer von ihm implementierten Klasse als „gleich“ erkannt werden sollen.
- Der Methode `equals(Object o)` eines Objekts A wird ein weiteres Objekt B übergeben, welches dann mit dem Objekt A verglichen werden kann (z.B. durch Vergleich der Attributwerte)

Kapitel 4: Fortgeschrittene Konzepte der Objektorientierung

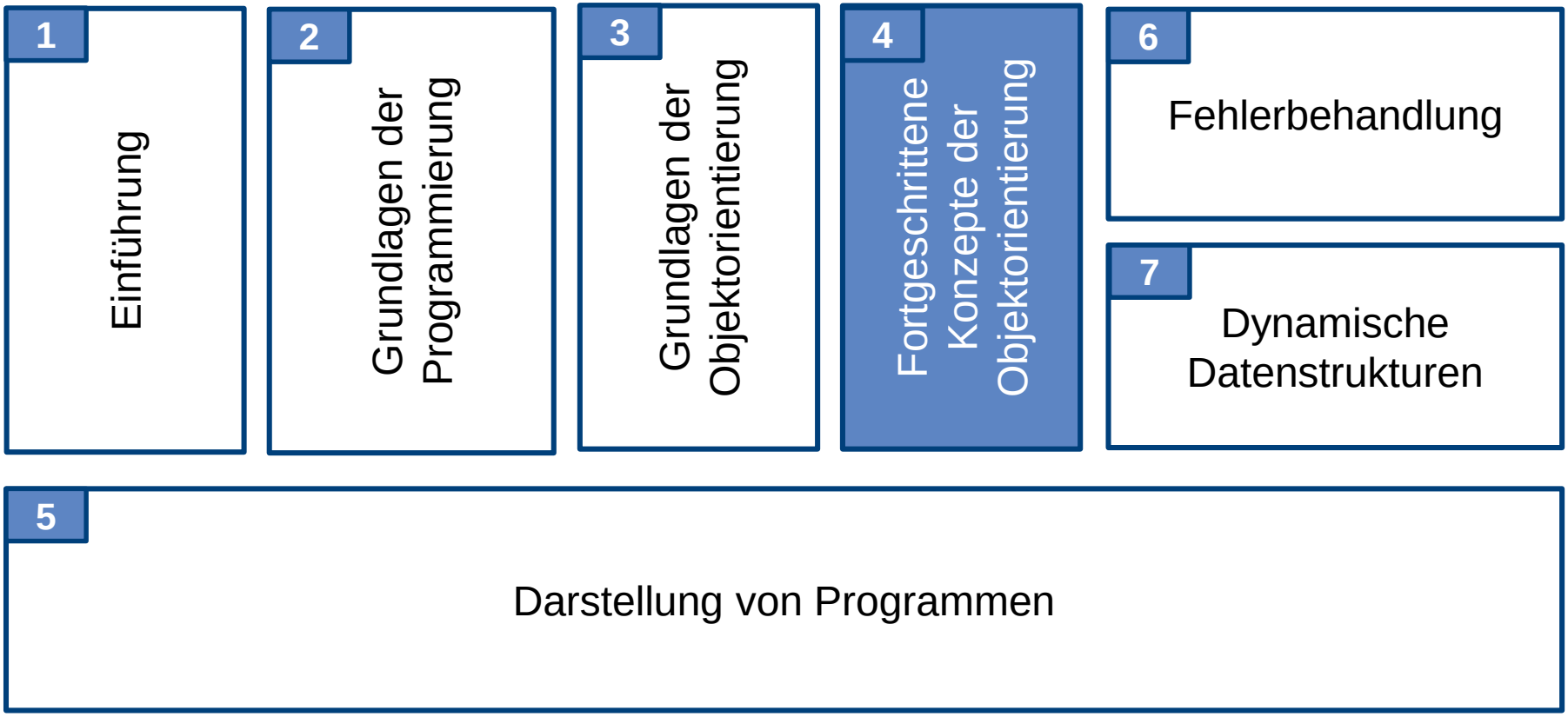
TECHNISCHE UNIVERSITÄT DARMSTADT

- Abstrakte Klassen in Java
- Garbage Collector: Löschen von Objekten
- Interfaces als Weg zur „Mehrfachvererbung“

Lernziele:

- Abstrakte Klassen in Java implementieren können und in Kombination mit dem Konzept des Polymorphismus anwenden können.
- Die Grundidee des Garbage Collectors verstehen und erklären können.
- Den Unterschied zwischen abstrakten Klassen und Interfaces kennen und erklären können.
- Interfaces in Java implementieren können und in Kombination mit dem Konzept des Polymorphismus anwenden können.

Thematische Übersicht



Einführung in abstrakte Klassen

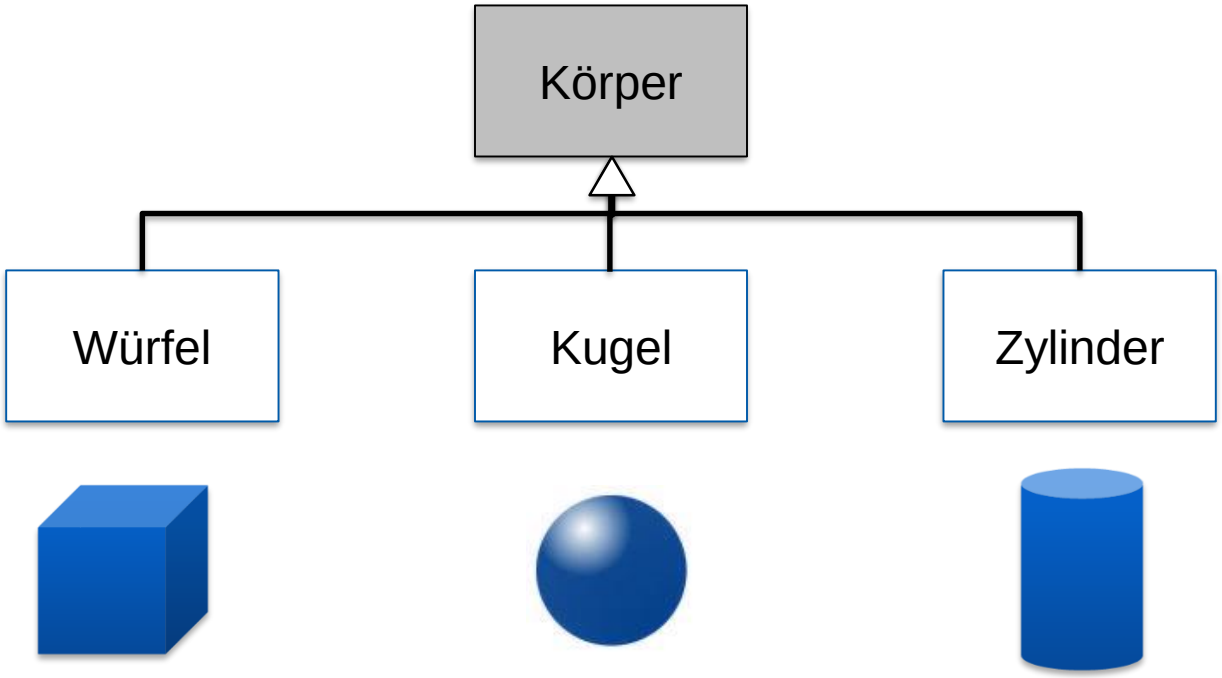
TECHNISCHE UNIVERSITÄT DARMSTADT

- Bisher haben wir mit **konkreten Klassen** gearbeitet
- **Abstrakte Klassen** unterscheiden sich von konkreten Klassen dadurch, dass von ihnen keine Objekte erzeugt werden können
- Es können aber Objekte von **erbenden (konkreten) Klassen** erzeugt werden – in einer Vererbungshierarchie kann die abstrakte Klasse als statischer Datentyp verwendet werden (Polymorphismus)
- Abstrakte Klassen deklarieren (oft) abstrakte Methoden: Eine **abstrakte Methode** hat keine Implementierung, nur der Methodenkopf ist festgelegt → muss in konkreter Unterklasse implementiert werden!

Einsatz abstrakter Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

Abstrakte Klassen werden oft als Ober-Klasse anstelle von konkreten Klassen verwendet, wenn Sub-Klassen teilweise gleiches Verhalten aufweisen, aber unterschiedliche Implementierungen erfordern, z. B.:



WIRTSCHAFTS INFORMATIK

Abstrakte Klassen in Java

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Abstrakte Klassen** werden durch das Schlüsselwort `abstract` kenntlich gemacht
- **Abstrakte Methoden** sind mit dem Modifier `abstract` deklariert und die Methodendeklaration endet mit einem Semikolon (anstelle eines Anweisungsblocks)
- **Hinweis**: Sobald in einer Klasse eine abstrakte Methode deklariert wird, muss auch die Klasse abstrakt deklariert werden.

```
public abstract class AbstrakteKlasse {

    //...

    public abstract void meineAbstrakteMethode();
}
```

Vorgehen bei Implementierung abstr. Klassen

TECHNISCHE UNIVERSITÄT DARMSTADT

- In der **abstrakten Ober-Klasse**:
  - Methoden, die in Unterklassen identisch realisiert sein sollen, werden als konkrete Methoden implementiert
  - Methoden, die in Unterklassen unterschiedlich realisiert sein sollen, werden als abstrakte Methoden deklariert
- In den **konkreten Sub-Klassen**:
  - Alle abstrakten Methoden aus der abstrakten Oberklasse werden implementiert (wird von Java erzwungen)
- **Hinweis**: Sub-Klassen, die von abstrakten Ober-Klassen **erben**, müssen **alle abstrakten Methoden implementieren oder wiederum abstrakt sein!**

Beispiel: Abstrakte Klassen (1/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die Ober-Klasse `Koerper` legt Methoden zur Berechnung von Volumen und Oberfläche fest, aber nicht wie diese implementiert werden
- Die Masse wird als Produkt aus Dichte und Volumen berechnet, hierzu kann bereits der Rückgabewert der abstrakten Methode `volumen()` verwendet werden

```
public abstract class Koerper {
    private double dichte;

    public Koerper(double dichte) { this.dichte = dichte; }

    public abstract double volumen();

    public abstract double oberflaeche();

    public double masse() { return dichte * volumen(); }
}
```

Beispiel: Abstrakte Klassen (2/2)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class Wuerfel extends Koerper  {

    private double kantenlaenge;

    public Wuerfel(double kantenlaenge, double dichte) {
        super(dichte);
        this.kantenlaenge_A = kantenlaenge;
    }

    public double oberflaeche() {
        return 6 * (kantenlaenge * kantenlaenge);
    }

    public double volumen() {
        return kantenlaenge * kantenlaenge * kantenlaenge;
    }
}
```

```
public class Kugel extends Koerper  {

    private double radius;

    public Kugel(double radius,
                 double dichte) {
        super(dichte);
        this.radius = radius;
    }

    public double oberflaeche() {
       return 4.0 * Math.PI *
                Math.pow(radius, 2);
    }

    public double volumen() {
       return 4.0/3.0 * Math.PI *
                Math.pow(radius, 3);
    }
}
```

Die Klassen `Würfel` und `Kugel` sind nicht abstrakt, daher müssen sie die in der Klasse `Koerper` abstrakt deklarierten Methoden implementieren.

WIRTSCHAFTS INFORMATIK

Welche Aufrufe funktionieren?

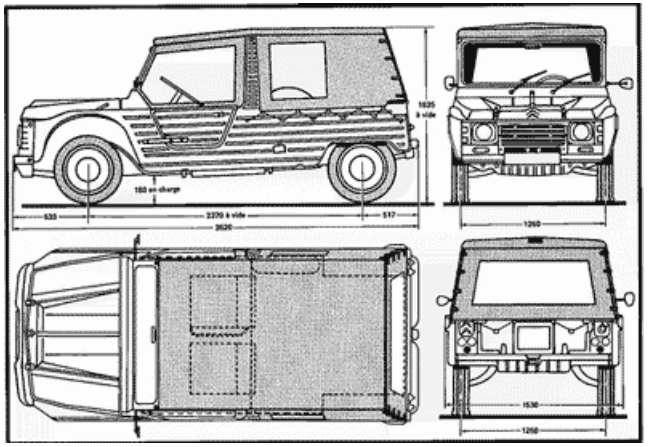

```
  public class MyClass {

    public static void main(String[] args) {
1     Kugel k = new Kugel(6.7, 5.0);
2     System.out.println(k.masse());

3     Koerper l = new Kugel(2.6, 7.0);
4     System.out.println(l.masse());

5     Koerper m = new Koerper(2.3);
6     System.out.println(m.masse());

7     Kugel n = new Koerper(4.2);
8     System.out.println(n.masse());


    }
  }
```

```
public class Kugel extends Koerper  {

    private double radius;

    public Kugel(double radius,
                 double dichte) {
        super(dichte);
        this.radius = radius;
    }

    public double volumen() {
       return 4.0/3.0 * Math.PI *
              Math.pow(radius, 3);
    }
}
```

```
public abstract class Koerper {
    private double dichte;

    public Koerper(double dichte) { this.dichte = dichte; }

    public abstract double volumen();

    public double masse() { return dichte * volumen(); }
}
```

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



Objekte löschen

Erweitert

TECHNISCHE UNIVERSITÄT DARMSTADT

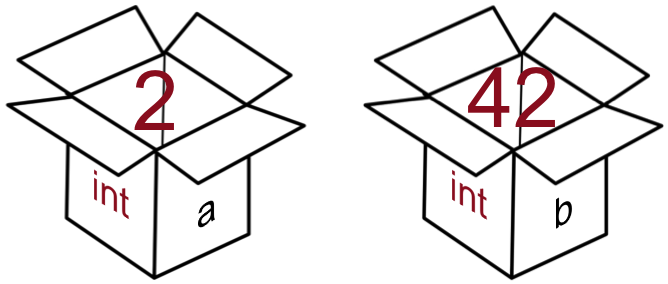
- Wie werden Objekte wieder gelöscht?
  - Vom Programmierer gar nicht
  - Nicht mehr benötigte (zugreifbare) Objekte werden von Java automatisch gelöscht („Garbage Collection“)

Der wesentliche Unterschied...

TECHNISCHE UNIVERSITÄT DARMSTADT

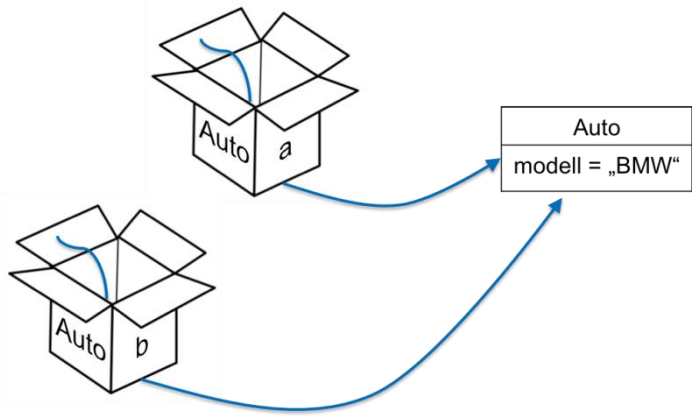
Variablen primitiven Datentyps:

```
int a = 1;
int b = 2;

a = b;
b = 42;
```



Variablen strukturierten Datentyps:

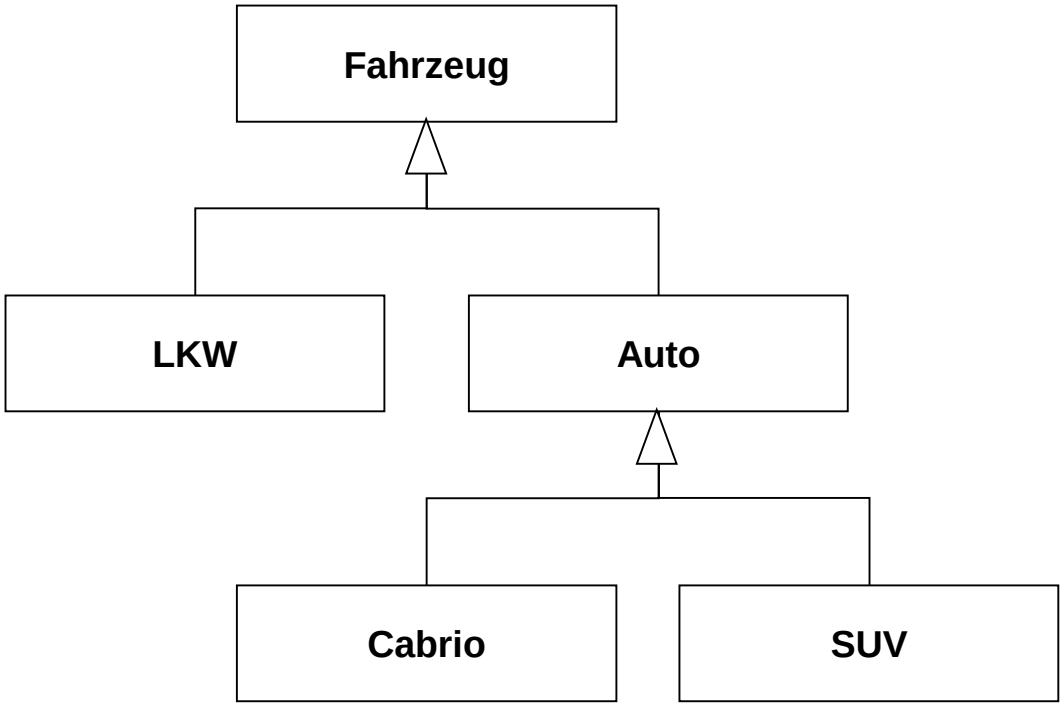
```
Auto a = new Auto();
a.modell ="Audi";
Auto b = new Auto();
b.modell="VW";

a = b;
b.modell="BMW";
```



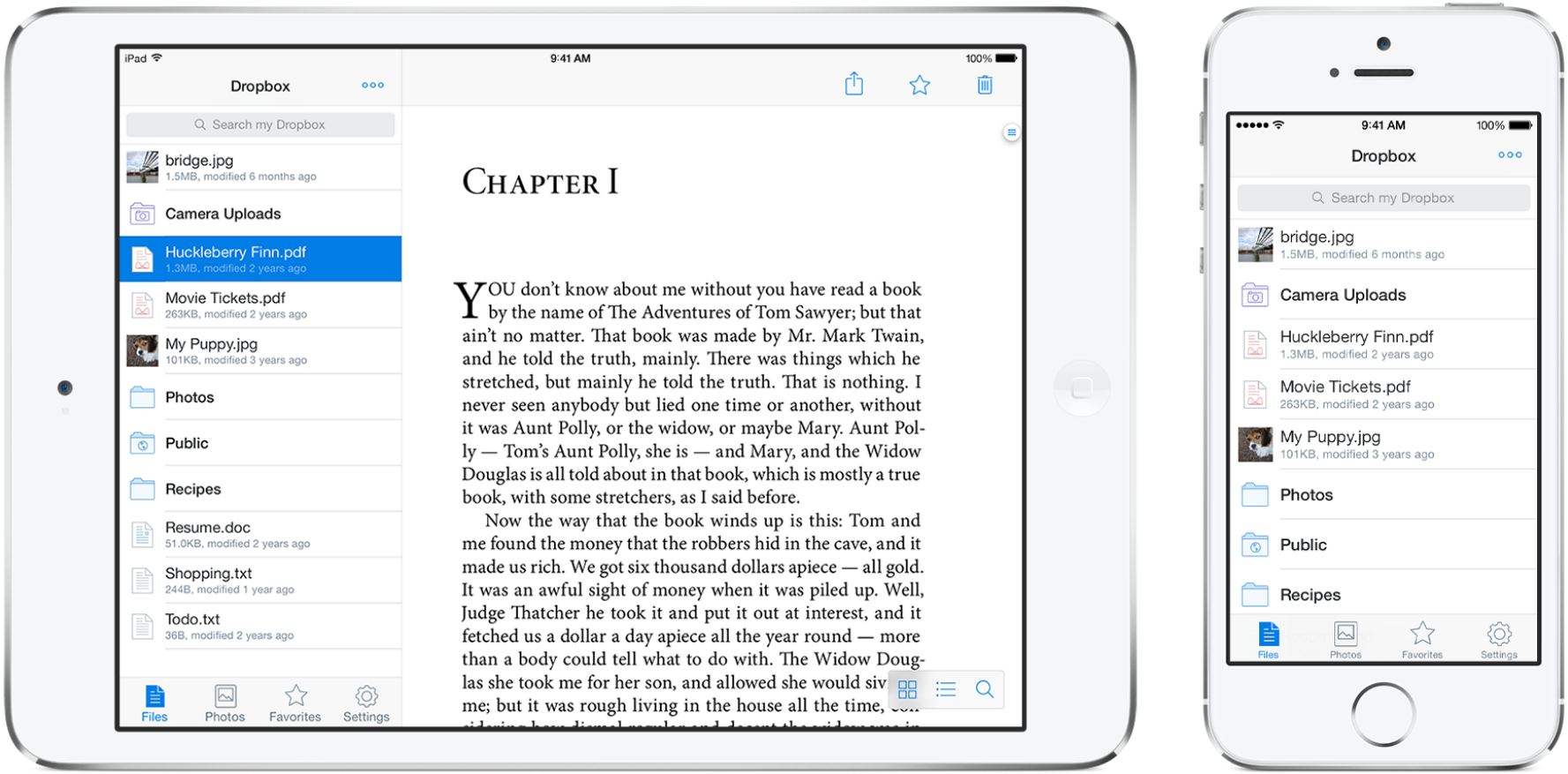
WIRTSCHAFTS INFORMATIK

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



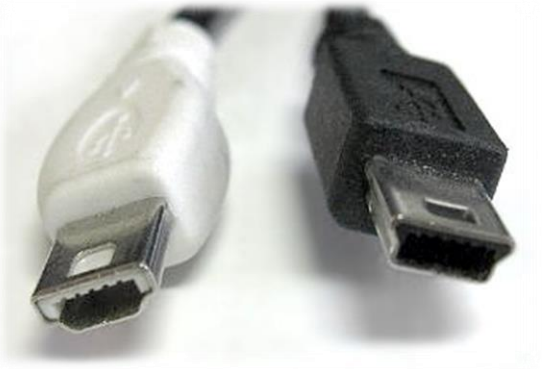
Interfaces (Beispiel)



Interfaces

Erweitert

TECHNISCHE UNIVERSITÄT DARMSTADT



- **Interfaces** enthalten Methoden-deklarationen, aber (grundsätzlich) keine Implementierungen dieser Methoden; diese Methoden werden in implementierenden Klassen implementiert
- Interfaces sind vergleichbar mit abstrakten Klassen, die nur
  - über Methoden verfügen, die implizit `public` und `abstract` sind und
  - über Attribute verfügen, die implizit `public` und `static` sind (die Attribute sind außerdem nicht veränderbar)
- Klassen können **beliebig viele Interfaces** implementieren. Interfaces ermöglichen eine Art **Mehrfachvererbung** in Java
- Weil Interfaces nur Methodendeklarationen enthalten, muss die implementierende Klasse alle Methoden des Interfaces implementieren
- In Verbindung mit Polymorphismus können Interfaces als statischer Datentyp verwendet werden

Interfaces in Java

Erweitert

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Interfaces** werden durch das Schlüsselwort `interface` deklariert

```
public interface ITest {
    void method();
}
```

- Methoden werden automatisch als `public abstract` deklariert, die Schlüsselworte `public abstract` sind redundant und sollen nicht angegeben werden

- Ein Interface kann andere Interfaces **erweitern**, es übernimmt die Methodendeklarationen aus allen übergeordneten Interfaces

Interfaces implementieren/erweitern

- Hinweis: Klassen können nur von einer Klasse erben, können aber beliebig viele Interfaces implementieren
- Implementieren Interfaces mit dem Schlüsselwort `implements`

```
public class MyClass extends ParentClass implements Itest1, Itest2 {
    public void method() { ... };
    ...
}
```

- Konkrete Klassen müssen alle im Interface deklarierten Methoden implementieren
- Interfaces können beliebig viele Interfaces erweitern, aber können nicht von Klassen erben

```
public interface INew extends ITest {
}
```

Beispiel: Interfaces (1/4)

- Ziel: Eine generische Klassen- und Interfacehierarchie zur Dateiverwaltung

- Beispielhafte Methoden, die für Dateien erlaubt sein können
  - Abspielen (z. B. Video oder Audio)
  - Teilen in sozialen Netzwerken (z. B. Mediendateien generell)
  - Durchsuchen (z. B. textbasierte Dateien)
  - …

Beispiel: Interfaces (2/4)

Erweitert

TECHNISCHE UNIVERSITÄT DARMSTADT

Klasse für Dateien:

```
public class Datei {
  //...
}
```

Interface für abspielbare Dateitypen:

```
public interface Abspielbar {
    void play();
}
```

Interface für teilbare Dateitypen:

```
public interface Teilbar {
    void share(String benutzer);
}
```

Beispiel: Interfaces (3/4)

Beispielklasse für Videodateien:

```
public class Videodatei extends Datei implements Abspielbar, Teilbar {

  //...

  public void play () {
    System.out.println(„Video wird abgespielt“);
   }


 public void share (String benutzer) {
    System.out.println(benutzer + „ hat ein Video angeschaut“);
    System.out.println(„>> Like, Share, or Comment <<“);
   }

}
```

Beispiel: Interfaces (4/4)

Beispiel für Verwendung mit Polymorphismus:

```
public class Mediaplayer {

  public static void main(String[] args) {

    Playable dateiEins = new Videodatei();
    dateiEins.play();

  }
}
```