Javakurs 2012 - Vererbung

Objektorientierte Programmierung II

Tim Jungnickel
by Mario Bodemann

9. März 2012

Inhaltsverzeichnis

Inhaltsverzeichnis

Was sind Objekte?

Objekte . . .

Was sind Objekte?

Objekte ...

- sind Instanzen von Klassen
- haben eine Identität, einen Zustand und ein Verhalten
- führen logisch zusammenhängenden Code zusammen
- ermöglichen auch komplexen Code zu verstehen und
- erlauben das Wiederverwenden von Code

Beispiel

Was passiert bei folgenden Programmfragmenten?

Listing 1: src/Human.java

```
1 public class Human{
2   private String name;
3   private int age;
4
5   public Human(){
6     this.name = "";
7     this.age = 0;
8   }
9 }
```

Listing 2: src/Human.java

```
1   public static void main(String[] args){
2     // create moe
3     Human moe = new Human();
4     moe.age = -12;
5   }
```

Beispiel

Was passiert bei folgenden Programmfragmenten?

Listing 3: src/Human.java

```
1 public class Human{
2   private String name;
3   private int age;
4
5   public Human(){
6     this.name = "";
7     this.age = 0;
8   }
9 }
```

Listing 4: src/Human.java

```
1   public static void main(String[] args){
2     // create moe
3     Human moe = new Human();
4     moe.age = -12;
5   }
```

- Eigenschaften ausserhalb des Wertebereichs (-12)
- Zugriff auf moe.age $= -12$ evtl. nicht möglich

Eine gekapselte Klasse

Listing 5: src/Human.java

```
1  public class Human{
2      private String name;
3      private int age;
4      public Human(String name, int age){
5          setName(name);
6          setAge(age);
7      }
8      public void setAge( int newAge ) {
9          // check if age is in range
10         if( newAge >= 0 && newAge < 130 ) {
11             this.age = newAge;
12         }
13     }
14 }
```

Richtiger Aufruf

Listing 6: src/Human.java

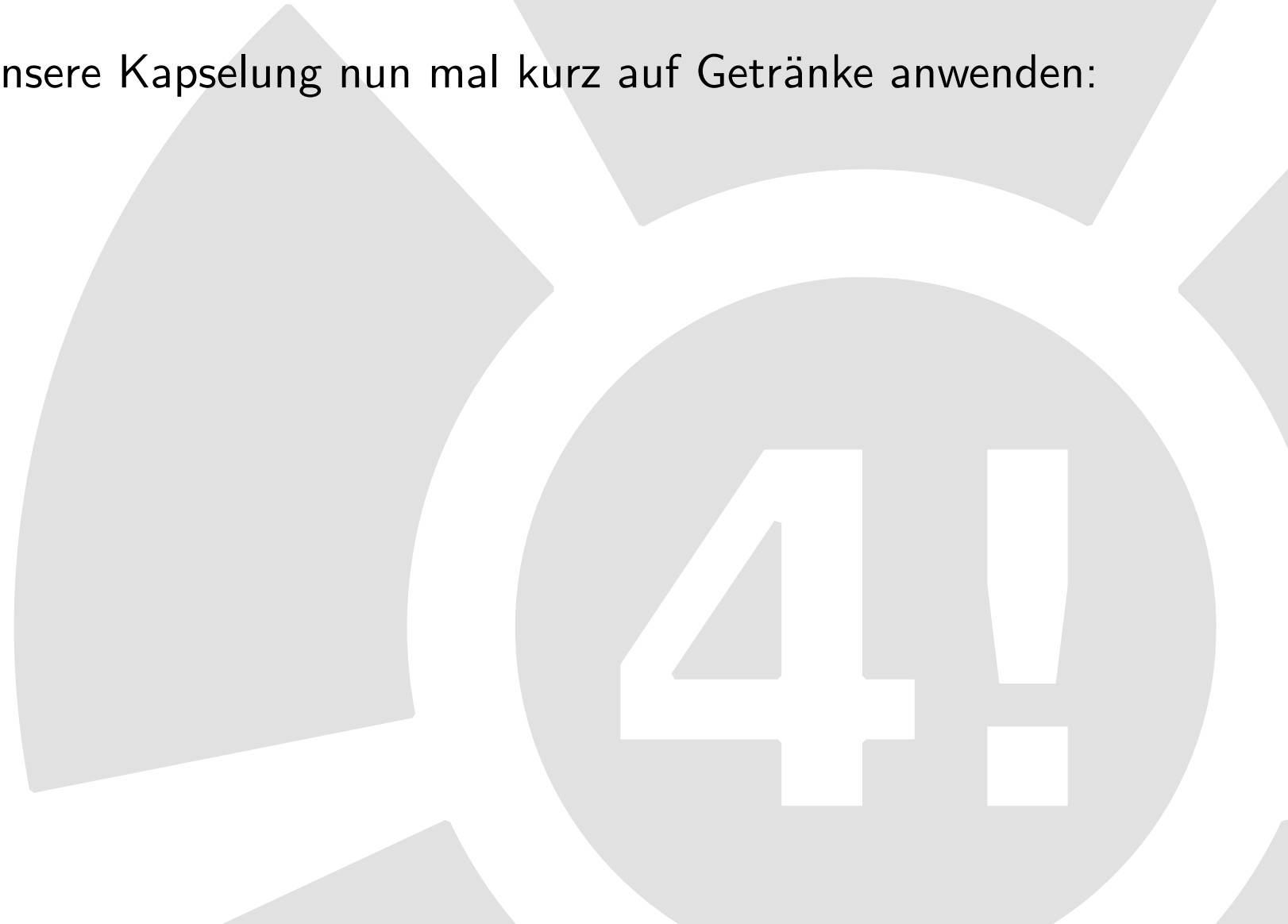
```
1  public static void main(String[] args){
2    // create moe
3    Human moe = new Human();
4    moe.setAge( -12 );
5  }
```

- Per Setter
- oder per Konstruktor
- keine nicht zugelassenen Stati möglich

Inhaltsverzeichnis

Getränke Beispiel

Wir wollen unsere Kapselung nun mal kurz auf Getränke anwenden:



Getränke Beispiel

Wir wollen unsere Kapselung nun mal kurz auf Getränke anwenden:

ColaDrink
-volume: double -name: String
+setVolume(newVolume:double) +getVolume(): double +setName() +getName(newName:String) +drink()

Getränke Beispiel

Wir wollen unsere Kapselung nun mal kurz auf Getränke anwenden:

ColaDrink
-volume: double -name: String
+setVolume(newVolume:double) +getVolume(): double +setName() +getName(newName:String) +drink()

FruitDrink
-volume: double -name: String
+setVolume(newVolume:double) +getVolume(): double +setName() +getName(newName:String) +drink()

Getränke Beispiel

Wir wollen unsere Kapselung nun mal kurz auf Getränke anwenden:

ColaDrink
-volume: double -name: String
+setVolume(newVolume:double) +getVolume(): double +setName() +getName(newName:String) +drink()

FruitDrink
-volume: double -name: String
+setVolume(newVolume:double) +getVolume(): double +setName() +getName(newName:String) +drink()

MateDrink
-volume: double -name: String
+setVolume(newVolume:double) +getVolume(): double +setName() +getName(newName:String) +drink()

Getränke Beispiel

Wir wollen unsere Kapselung nun mal kurz auf Getränke anwenden:

ColaDrink
-volume: double -name: String
+setVolume(newVolume:double) +getVolume(): double +setName() +getName(newName:String) +drink()

FruitDrink
-volume: double -name: String
+setVolume(newVolume:double) +getVolume(): double +setName() +getName(newName:String) +drink()

MateDrink
-volume: double -name: String
+setVolume(newVolume:double) +getVolume(): double +setName() +getName(newName:String) +drink()

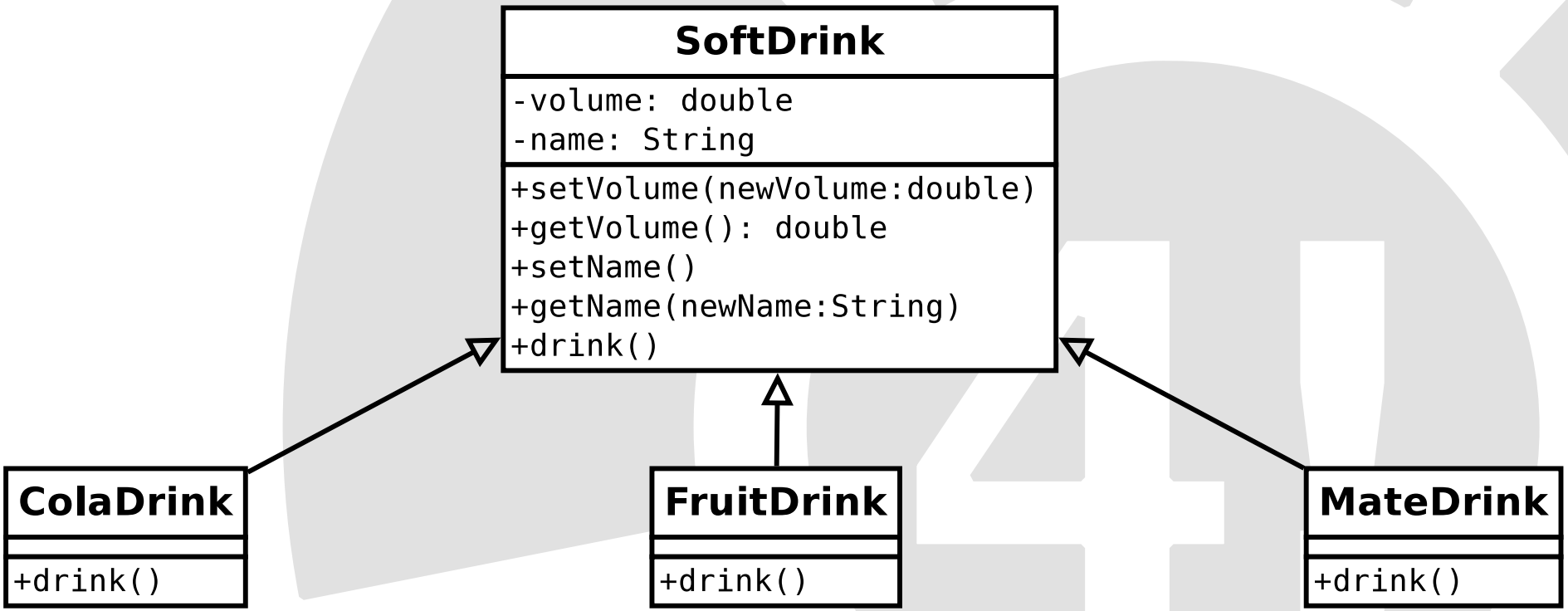
- Alle Getränkesorten haben Eigenschaften und Verhalten gleich
- Alle teilen sich die Getter und Setter der Eigenschaften
- Unterschiede
  - den Geschmack
  - Verhalten und Eigenschaften

Getränke Lösung

- Schaffung einer neuen Klasse, die die Eigenschaften kombiniert
- Dies nennt man Vererbung und kann so dargestellt werden:

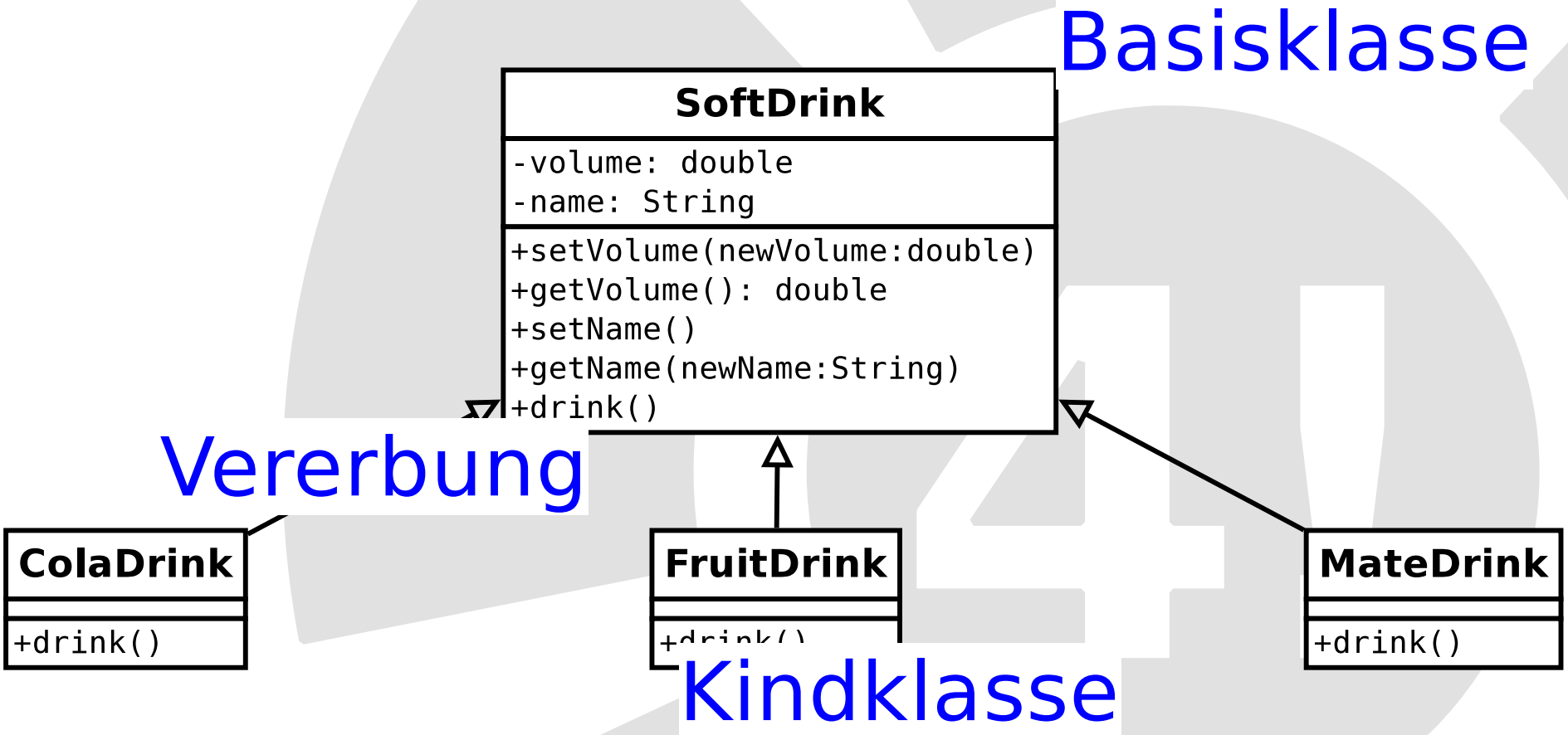
Getränke Lösung

- Schaffung einer neuen Klasse, die die Eigenschaften kombiniert
- Dies nennt man Vererbung und kann so dargestellt werden:



Getränke Lösung - mit Beschreibung

- Schaffung einer neuen Klasse, die die Eigenschaften kombiniert
- Dies nennt man Vererbung und kann so dargestellt werden:



SoftDrink Klasse: Die Basisklasse Auszug

Listing 7: src/SoftDrinks.java

```
1  class SoftDrink {
2      private double volume;
3      private String name;
4
5      public SoftDrink() {
6          this.volume = 0.0;
7          this.name = "Default Drink";
8      }
9      public void drink() {
10         System.out.println("You just drank a default drink!");
11     }
```

ColaDrink: Eine abgeleitete Klasse

Listing 8: src/SoftDrinks.java

```
1 class ColaDrink extends SoftDrink {
2     public void drink() {
3         System.out.println("You just drank a tasty cola drink!");
4     }
5 }
```

ColaDrink: Eine abgeleitete Klasse

Listing 9: src/SoftDrinks.java

```
1 class ColaDrink extends SoftDrink {
2   public void drink() {
3     System.out.println("You just drank a tasty cola drink!");
4   }
5 }
```

extends

- Sorgt dafür, dass die Klasse **ColaDrink** von **SoftDrink** erbt
- Referenziert alle Methoden und Klassen der **Basisklasse**
- Alles was nicht in **ColaDrink** steht wird in **SoftDrink** gesucht und benutzt

Benutzung von ColaDrink Klasse

Listing 10: src/SoftDrinks.java

```
1    ColaDrink cola = new ColaDrink();
2    cola.setVolume( 1.0 );
3    cola.setName( "Buzz Cola" );
4    System.out.println( "Name: " + cola.getName());
5    cola.drink();
```

Benutzung von ColaDrink Klasse

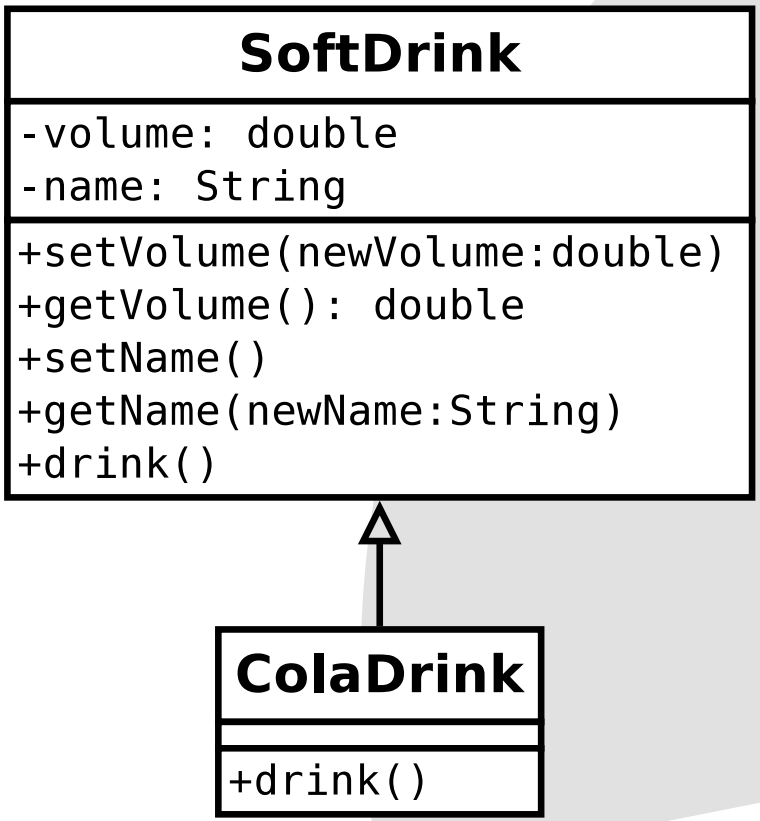
Listing 11: src/SoftDrinks.java

```
1    ColaDrink cola = new ColaDrink();
2    cola.setVolume( 1.0 );
3    cola.setName( "Buzz Cola" );
4    System.out.println( "Name: " + cola.getName());
5    cola.drink();
```

Ausgabe

```
Name: Buzz Cola
You just drank a tasty cola drink!
```

Was passiert bei **cola.setVolume(1.0);**?

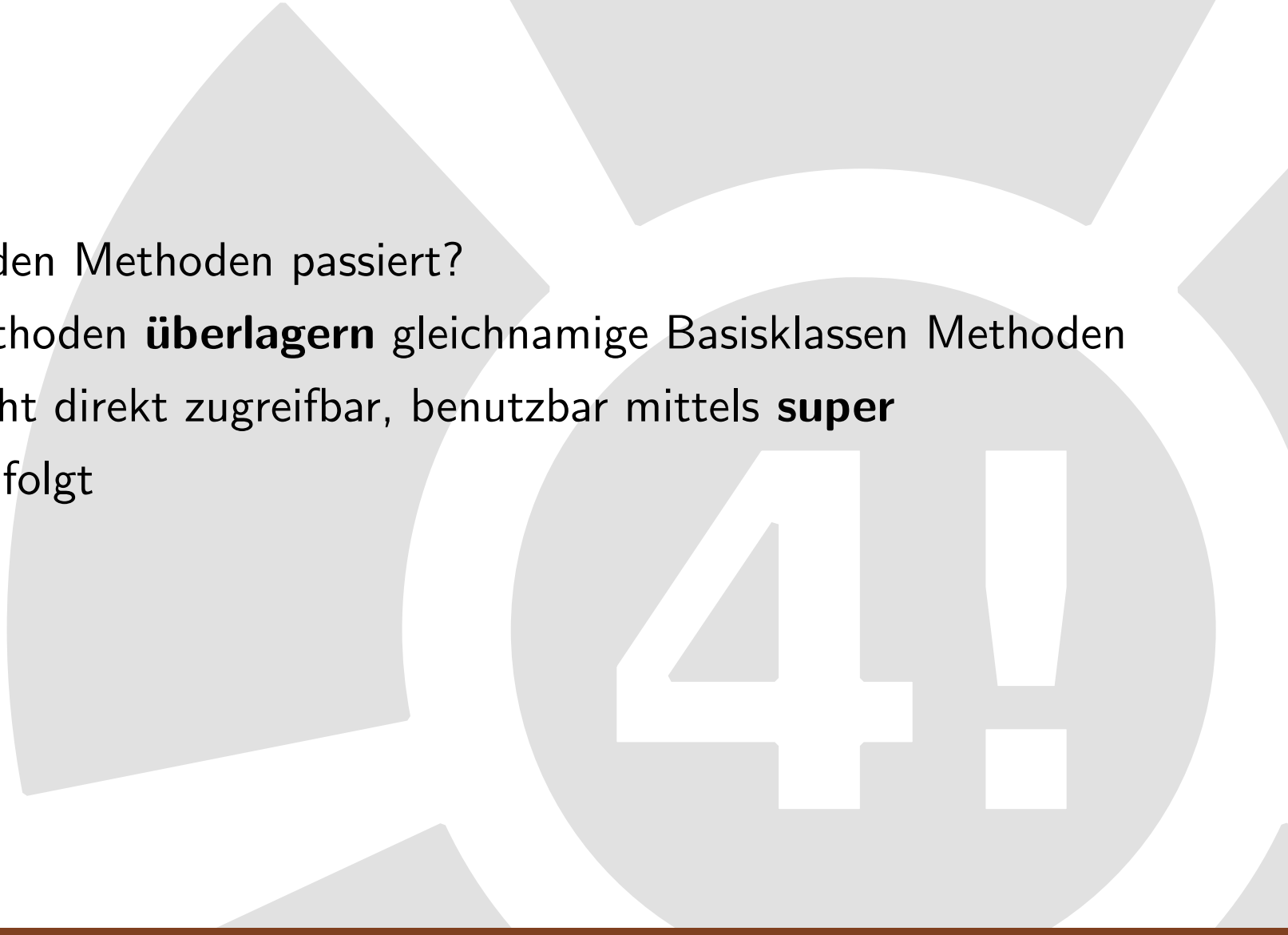


1. VM findet die Klasse von cola: **ColaDrink**
2. Sucht in der Klasse die Methode **setVolume(..)** findet sie nicht
3. Sucht weiter in der nächster Basisklasse von ColaDrink: **SoftDrink**
4. Methode gefunden
5. Sonst: Compile-Zeitfehler wenn Methode nicht gefunden.

Wo kommt *setVolume(..)* her?

Was ist mit den Methoden passiert?

- Alle Methoden **überlagern** gleichnamige Basisklassen Methoden
- Sind nicht direkt zugreifbar, benutzbar mittels **super**
- Beispiel folgt



Aufruf von Basisklassen Methoden

Auszug aus MateDrink:

Listing 12: src/SoftDrinks.java

```
1  public void drink() {
2    super.drink();
3    System.out.println( "You are now filled with powerfull energy!" );
4  }
```

Aufruf von Basisklassen Methoden

Auszug aus MateDrink:

Listing 13: src/SoftDrinks.java

```
1  public void drink() {
2    super.drink();
3    System.out.println( "You are now filled with powerfull energy!" );
4  }
```

super

- Ist das Objekt der Basisklasse
- Zugriff auf überlagerte Methoden möglich
- *this.drink() == drink()* (rekursiver Aufruf)
- super.drink() ruft immer SoftDrink.drink() auf
- gleiches gilt auch für Eigenschaften

Aufruf von Basisklassen Methoden

Listing 14: src/SoftDrinks.java

```
1  public void drink() {
2    super.drink();
3    System.out.println( "You are now filled with powerfull energy!" );
4  }
```

Listing 15: src/SoftDrinks.java

```
1    // create a mate to show super in methods
2    MateDrink mate = new MateDrink();
3    mate.drink();
```

Aufruf von Basisklassen Methoden

Listing 16: src/SoftDrinks.java

```
1  public void drink() {
2    super.drink();
3    System.out.println( "You are now filled with powerfull energy!" );
4  }
```

Listing 17: src/SoftDrinks.java

```
1    // create a mate to show super in methods
2    MateDrink mate = new MateDrink();
3    mate.drink();
```

Ausgabe

```
You just drank a default drink!
You are now filled with powerfull energy!
```

Inhaltsverzeichnis

Warum das Ganze?

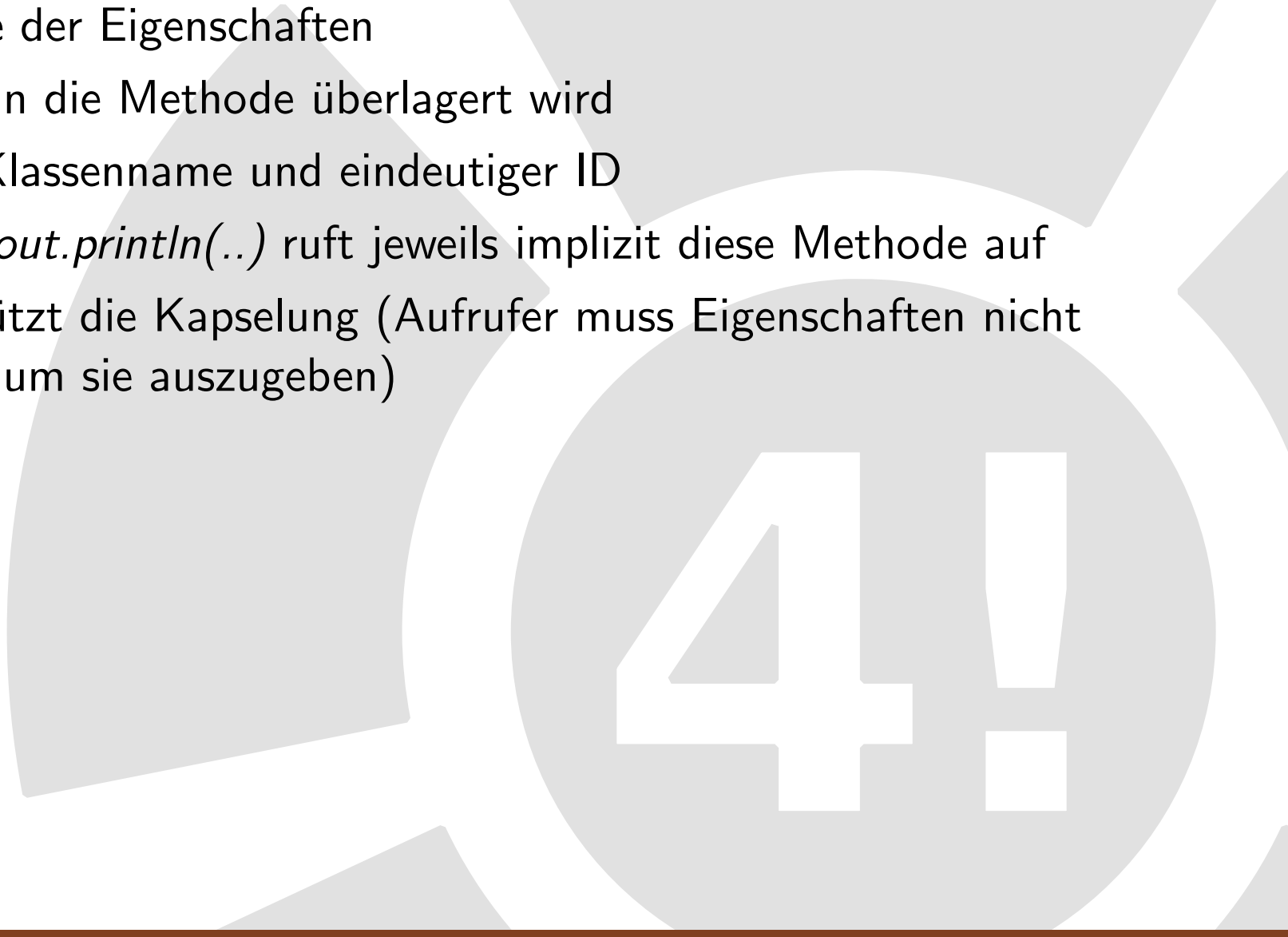
- Vererbung vereinfacht Handhabung von komplexen Systemen
- erhöht Wiederverwendbarkeit
- Zusätzliche Vereinheitlichung aller Klassen:
  - *Jede* Klasse erbt von **Object** (außer Object selber ;-))
  - Auch die die nicht *explizit* erben (bspw. **Human**, **SoftDrink**, ...)
  - Jedes Objekt kann als **Object** interpretiert werden
  - Jedes Objekt bietet ein Mindestmaß an Funktionalitäten

Methoden von Object

- *Object* bietet folgende Methoden an
  - **public String toString()**
  - **public boolean equals(Object obj)**
  - public Class getClass()
  - public int hashCode()
  - protected Object clone()
  - public void notify()
  - public void notifyAll()
  - public void wait(long arg0)
  - public void wait(long timeout, int nanos)
  - public void wait()
  - protected void finalize()

toString - Theorie

- Ausgabe der Eigenschaften
- Nur wenn die Methode überlagert wird
- Sonst: Klassenname und eindeutiger ID
- *System.out.println(..)* ruft jeweils implizit diese Methode auf
- Unterstützt die Kapselung (Aufrufer muss Eigenschaften nicht kennen, um sie auszugeben)



toString - Theorie

- Ausgabe der Eigenschaften
- Nur wenn die Methode überlagert wird
- Sonst: Klassenname und eindeutiger ID
- *System.out.println(..)* ruft jeweils implizit diese Methode auf
- Unterstützt die Kapselung (Aufrufer muss Eigenschaften nicht kennen, um sie auszugeben)

Beispielmethode in SoftDrink:

Listing 19: src/SoftDrinks.java

```
1 class MateDrink extends SoftDrink {
2     public String toString() {
3         return super.toString() + " (of powerfull energy)";
4     }
5 }
```

toString - Praxis

Listing 20: src/SoftDrinks.java

```
1  MateDrink mate = new MateDrink();
2  ColaDrink cola = new ColaDrink();
3
4  // output two objects
5  System.out.println( mate );
6  System.out.println( cola );
```

toString - Praxis

Listing 21: src/SoftDrinks.java

```
1     MateDrink mate = new MateDrink();
2     ColaDrink cola = new ColaDrink();
3
4     // output two objects
5     System.out.println( mate );
6     System.out.println( cola );
```

Ausgabe

```
Softdrink named Default Drink, holding 0.0l.
        (of powerfull energy)
Softdrink named Default Drink, holding 0.0l.
```

Vergleich zwischen Mate und Cola

Vergleich zwischen Mate und Cola

Listing 23: src/SoftDrinks.java

```
1     if( (Object)mate == (Object)cola ) {
2       System.out.println("Mate and Cola are the same!");
3     } else {
4       System.out.println("Indeed, Mate and Cola are different!");
5     }
```

Vergleich zwischen Mate und Cola

Listing 24: src/SoftDrinks.java

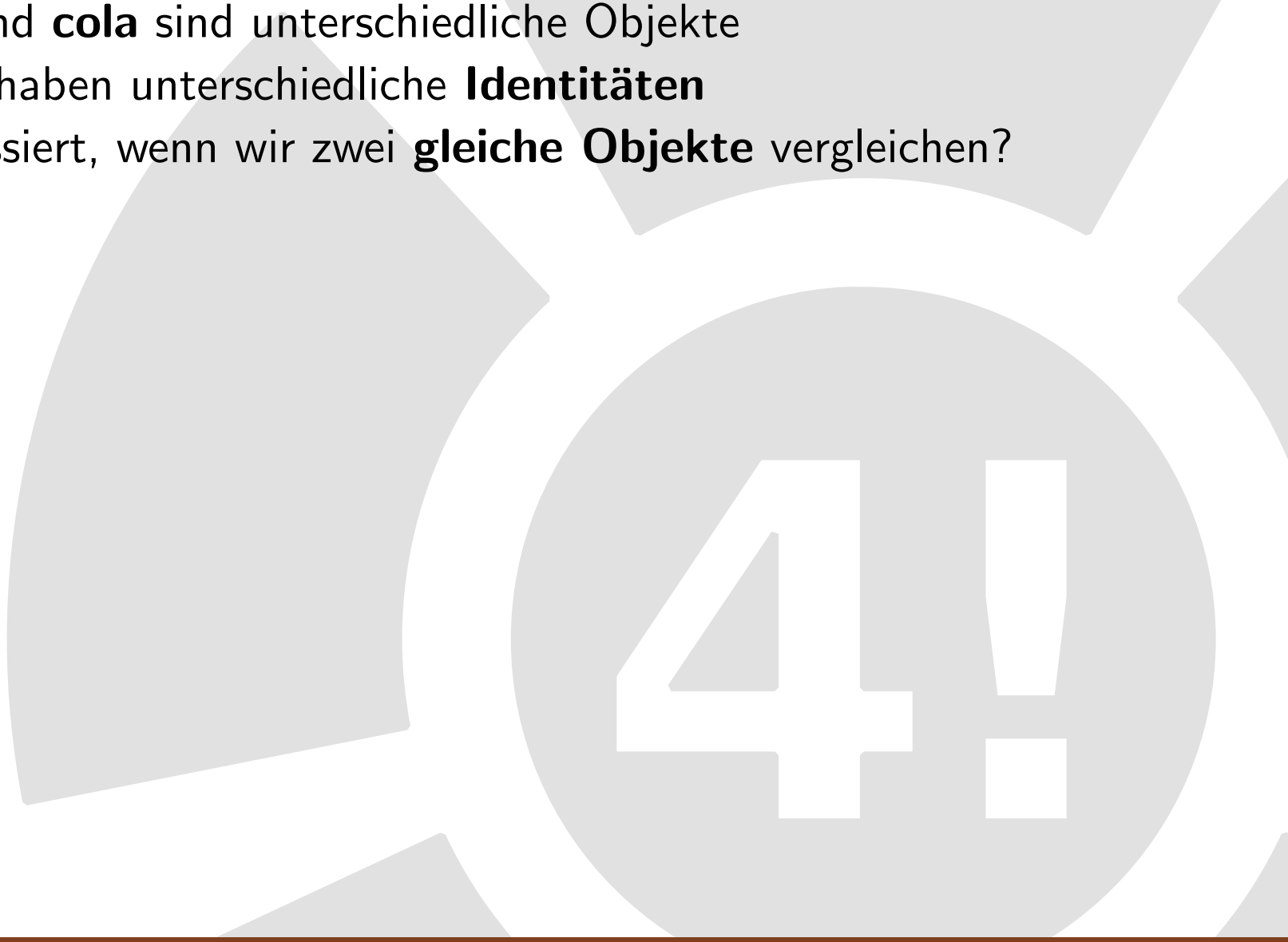
```
1     if( (Object)mate == (Object)cola ) {
2       System.out.println("Mate and Cola are the same!");
3     } else {
4       System.out.println("Indeed, Mate and Cola are different!");
5     }
```

Ausgabe

```
Indeed, Mate and Cola are different!
```

Warum sind Cola und Mate unterschiedlich?

- **mate** und **cola** sind unterschiedliche Objekte
- d.h. sie haben unterschiedliche **Identitäten**
- Was passiert, wenn wir zwei **gleiche Objekte** vergleichen?



Warum sind Cola und Mate unterschiedlich?

- **mate** und **cola** sind unterschiedliche Objekte
- d.h. sie haben unterschiedliche **Identitäten**
- Was passiert, wenn wir zwei **gleiche Objekte** vergleichen?

Listing 26: src/SoftDrinks.java

```
1  MateDrink yourMate = new MateDrink();
2  if( mate == yourMate ) {
3    System.out.println("Your and mine mate are the same!");
4  } else {
5    System.out.println("Arrr, you have an other mate than me.");
6  }
7
```

Warum sind Cola und Mate unterschiedlich?

- **mate** und **cola** sind unterschiedliche Objekte
- d.h. sie haben unterschiedliche **Identitäten**
- Was passiert, wenn wir zwei **gleiche Objekte** vergleichen?

Listing 27: src/SoftDrinks.java

```
1  MateDrink yourMate = new MateDrink();
2  if( mate == yourMate ) {
3    System.out.println("Your and mine mate are the same!");
4  } else {
5    System.out.println("Arrr, you have an other mate than me.");
6  }
7
```

Ausgabe

```
Arrr, you have an other mate than me.
```

Zwei fast gleiche Objekte

- Vergleich sollte sich auf die *Attribute* (*name*, *age*) beziehen
- Verglichen wurden aber die Referenzen
- Unterschiedliche Referenzen auf unterschiedliche Objekte mit gleichen Werten
- Also: Vergleichsmethode für Wertgleichheit benutzen: equals

equals-Methode

Listing 28: src/SoftDrinks.java

```
1   public boolean equals(Object other) {
2     if( other instanceof SoftDrink ) {
3       SoftDrink otherDrink = (SoftDrink)other;
4       return otherDrink.getName().equals(this.getName()) &&
5         otherDrink.getVolume() == this.getVolume();
6     } else {
7       return false;
8     }
9   }
10 }
```

equals-Methode

Listing 29: src/SoftDrinks.java

```
1   public boolean equals(Object other) {
2     if( other instanceof SoftDrink ) {
3       SoftDrink otherDrink = (SoftDrink)other;
4       return otherDrink.getName().equals(this.getName()) &&
5         otherDrink.getVolume() == this.getVolume();
6     } else {
7       return false;
8     }
9   }
10 }
```

- Zuerst wird überprüft, ob *other* vom gleichen Typ (*SoftDrink*) ist
- SoftDrink otherDrink = (SoftDrink)other; Wandelt ein *Object* in einen *SoftDrink* um (**Typecasting**)
- Nun ist der Wertevergleich mittels gettern möglich.

Wertevergleich zwischen meiner und deiner Mate

Listing 30: src/SoftDrinks.java

```
1    if( mate.equals(yourMate) == true ) {
2      System.out.println("Our mate soft drinks are the same");
3    } else {
4      System.out.println("We do have different mate drinks!");
5    }
```

Wertevergleich zwischen meiner und deiner Mate

Listing 31: src/SoftDrinks.java

```
1     if( mate.equals(yourMate) == true ) {
2       System.out.println("Our mate soft drinks are the same");
3     } else {
4       System.out.println("We do have different mate drinks!");
5     }
```

Ausgabe

```
Our mate soft drinks are the same
```

Inhaltsverzeichnis

Beispiel

Was passiert bei folgenden Programmfragmenten?

Listing 32: src/Human.java

```
1 public class Human{
2   private String name;
3   private int age;
4
5   public Human(){
6     this.name = "";
7     this.age = 0;
8   }
9 }
```

Listing 33: src/Human.java

```
1   public static void main(String[] args){
2     // create moe
3     Human moe = new Human();
4     moe.age = -12;
5   }
```

Beispiel

Was passiert bei folgenden Programmfragmenten?

Listing 34: src/Human.java

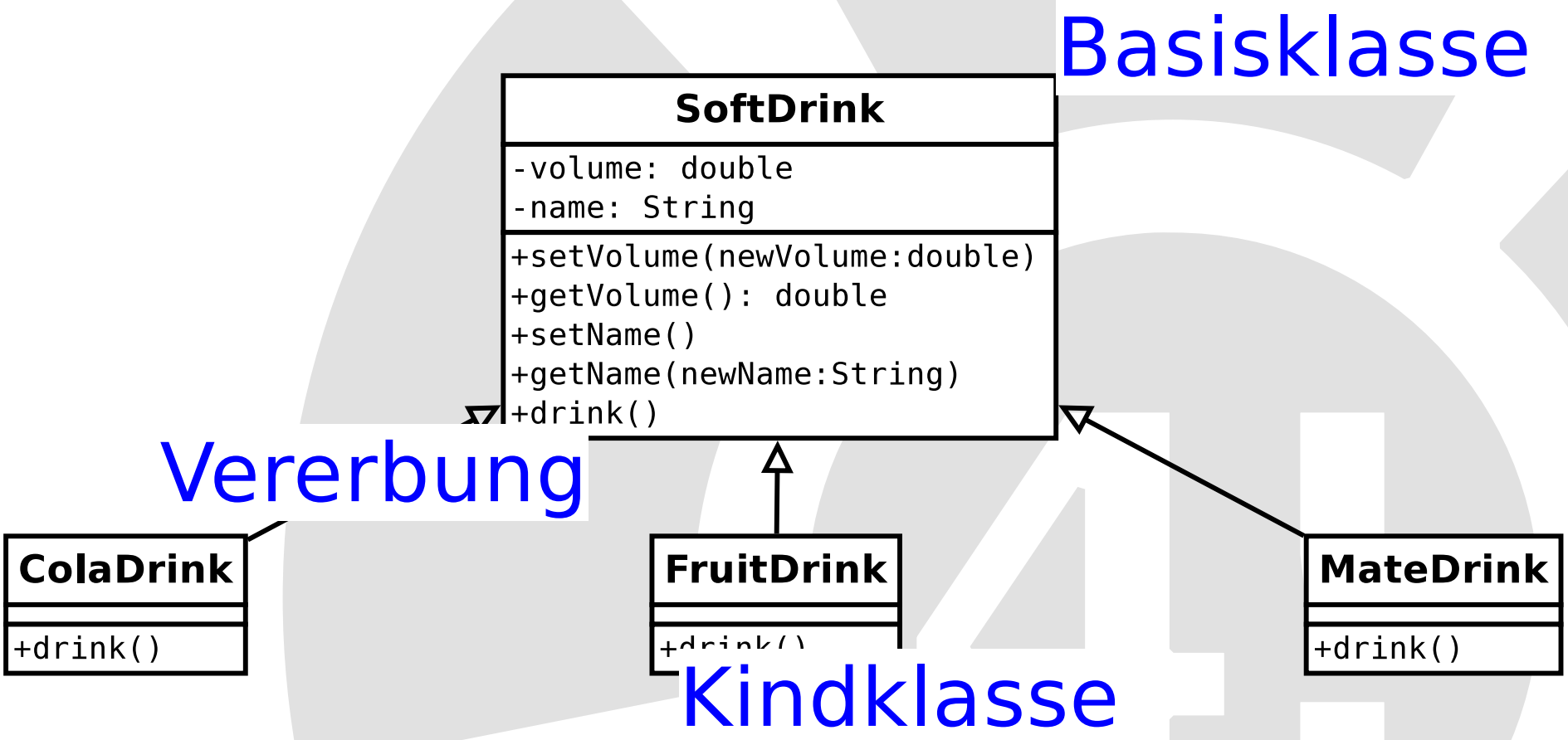
```
1 public class Human{
2   private String name;
3   private int age;
4
5   public Human(){
6     this.name = "";
7     this.age = 0;
8   }
9 }
```

Listing 35: src/Human.java

```
1   public static void main(String[] args){
2     // create moe
3     Human moe = new Human();
4     moe.age = -12;
5   }
```

- Eigenschaften ausserhalb des Wertebereichs (-12)
- Zugriff auf moe.age $= -12$ evtl. nicht möglich

Klassenhirarchie



Methoden von Object

- *Object* bietet folgende Methoden an
  - **public String toString()**
  - **public boolean equals(Object obj)**
  - public Class getClass()
  - public int hashCode()
  - protected Object clone()
  - public void notify()
  - public void notifyAll()
  - public void wait(long arg0)
  - public void wait(long timeout, int nanos)
  - public void wait()
  - protected void finalize()

Weitere Themen von Interesse

- Warum und wie funktioniert *(SoftDrink)other*?
  - Schlagworte: Polymorphie, Typecasting
- Mehr über Objektorientierung
  - Schlagworte: Interfaces, abstrakte Klassen, Packages
- Algorithmen über Typen statt über Werte/Variablen
  - Sortieren ohne Typinformation
  - Schlagwort: Generics
- Entwurfsmuster
- Literatur
  - `http://openbook.galileocomputing.de/javainsel/`