

Javakurs 2011 - Vererbung


Objektorientierte Programmierung II

Mario Bodemann

21. März 2011



This work is licensed under the *Creative Commons Attribution-ShareAlike 3.0 License*.

- 
- 1 Wiederholung
 - 2 Vererbung
 - Getränke
 - Klassenhierarchie
 - Zugriff auf Basisklasse
 - 3 Standard-Methoden
 - Object
 - toString
 - equals
 - 4 Zusammenfassung
 - Lessons learned
 - Ausblick
 - Fragen

Inhaltsverzeichnis

1 Wiederholung

2 Vererbung

- Getränke
- Klassenhierarchie
- Zugriff auf Basisklasse

3 Standard-Methoden

- Object
- toString
- equals

4 Zusammenfassung

- Lessons learned
- Ausblick
- Fragen



4!

Was sind Objekte?

Objekte ...



4!

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

4!

Beispiel

Was passiert bei folgenden Programmfragmenten?

Listing 1: src/Human.java

```
4 public class Human{  
5     private String name;  
6     private int age;  
7  
8     public Human(){  
9         this.name = "";  
10        this.age = 0;  
11    }  
92 }
```

Listing 2: src/Human.java

```
53 public static void main(String[] args){  
54     // create moe  
55     Human moe = new Human();  
56     moe.age = -12;  
91 }
```

Beispiel

Was passiert bei folgenden Programmfragmenten?

Listing 3: src/Human.java

```
4 public class Human{  
5     private String name;  
6     private int age;  
7  
8     public Human(){  
9         this.name = "";  
10        this.age = 0;  
11    }  
92 }
```

Listing 4: src/Human.java

```
53 public static void main(String[] args){  
54     // create moe  
55     Human moe = new Human();  
56     moe.age = -12;  
91 }
```

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

Eine gekapselte Klasse

Listing 5: src/Human.java

```
4 public class Human{
5     private String name;
6     private int age;
13    public Human(String name, int age){
14        setName(name);
15        setAge(age);
16    }
24    public void setAge( int newAge ) {
25        // check if age is in range
26        if( newAge >= 0 && newAge < 130 ) {
27            this.age = newAge;
28        }
29    }
92 }
```


Richtiger Aufruf

Listing 6: src/Human.java

```
53 public static void main(String[] args){  
54     // create moe  
55     Human moe = new Human();  
57     moe.setAge( -12 );  
91 }
```

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

Inhaltsverzeichnis

1 Wiederholung

2 Vererbung

- Getränke
- Klassenhierarchie
- Zugriff auf Basisklasse

3 Standard-Methoden

- Object
- toString
- equals

4 Zusammenfassung

- Lessons learned
- Ausblick
- Fragen



Getränke Beispiel

Wir wollen unsere Kapselung nun mal kurz auf Getränke anwenden:



4!

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()

4!

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()
```

4!

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

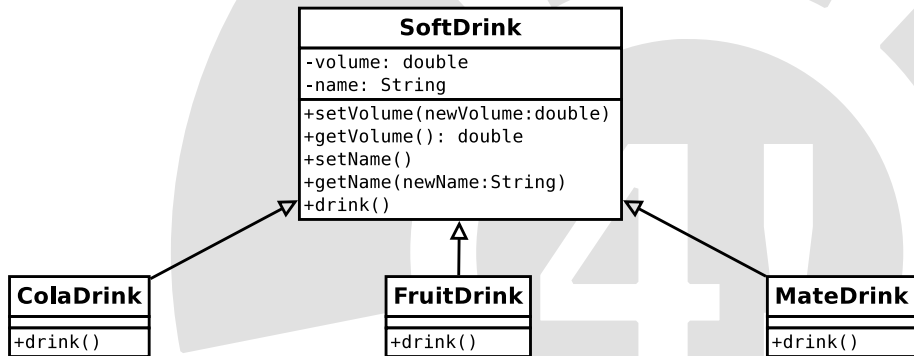
- Schaffung einer neuen Klasse, die die Eigenschaften kombiniert
- Dies nennt man Vererbung und kann so dargestellt werden:



4!

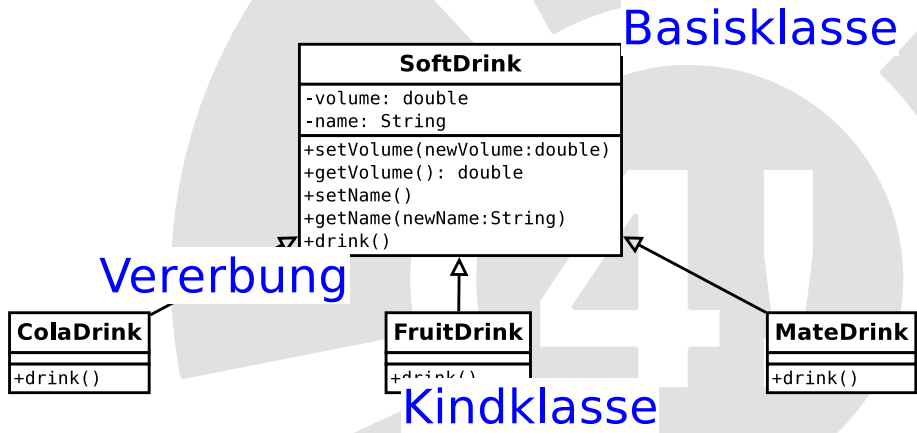
Getränke Lösung

- 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

```
78 class SoftDrink {  
79     private double volume;  
80     private String name;  
81  
82     public SoftDrink() {  
83         this.volume = 0.0;  
84         this.name = "Default Drink";  
85     }  
09     public void drink() {  
10         System.out.println("You just drank a default drink!");  
11     }
```

ColaDrink: Eine abgeleitete Klasse

Listing 8: src/SoftDrinks.java

```
47 class ColaDrink extends SoftDrink {  
48     public void drink() {  
49         System.out.println("You just drank a tasty cola drink!");  
50     }  
51 }
```

ColaDrink: Eine abgeleitete Klasse

Listing 9: src/SoftDrinks.java

```
47 class ColaDrink extends SoftDrink {  
48     public void drink() {  
49         System.out.println("You just drank a tasty cola drink!");  
50     }  
51 }
```

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

```
10 ColaDrink cola = new ColaDrink();  
11 cola.setVolume( 1.0 );  
12 cola.setName( "Buzz Cola" );  
13 System.out.println( "Name: " + cola.getName());  
14 cola.drink();
```

Benutzung von ColaDrink Klasse

Listing 11: src/SoftDrinks.java

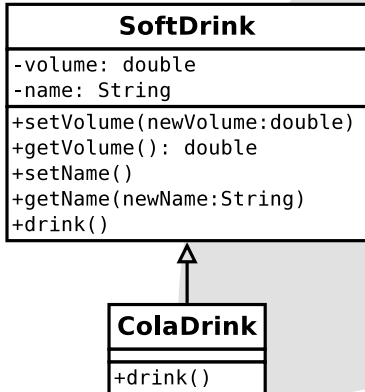
```
10 ColaDrink cola = new ColaDrink();  
11 cola.setVolume( 1.0 );  
12 cola.setName( "Buzz Cola" );  
13 System.out.println( "Name: " + cola.getName());  
14 cola.drink();
```

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



4!

Aufruf von Basisklassen Methoden

Listing 12: src/SoftDrinks.java

```
66 public void drink() {  
67     super.drink();  
68     System.out.println( "You are now filled with powerfull energy!" );  
69 }
```

4!

Aufruf von Basisklassen Methoden

Listing 13: src/SoftDrinks.java

```
66 public void drink() {  
67     super.drink();  
68     System.out.println( "You are now filled with powerfull energy!" );  
69 }
```

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

```
66 public void drink() {  
67     super.drink();  
68     System.out.println( "You are now filled with powerfull energy!" );  
69 }
```

Listing 15: src/SoftDrinks.java

```
6 // create a mate to show super in methods  
7 MateDrink mate = new MateDrink();  
8 mate.drink();
```

Aufruf von Basisklassen Methoden

Listing 16: src/SoftDrinks.java

```
66 public void drink() {  
67     super.drink();  
68     System.out.println( "You are now filled with powerfull energy!" );  
69 }
```

Listing 17: src/SoftDrinks.java


```
6 // create a mate to show super in methods  
7 MateDrink mate = new MateDrink();  
8 mate.drink();
```

Ausgabe

You just drank a default drink!

You are now filled with powerfull energy!

Inhaltsverzeichnis

- 
- 1 Wiederholung
 - 2 Vererbung
 - Getränke
 - Klassenhierarchie
 - Zugriff auf Basisklasse
 - 3 **Standard-Methoden**
 - Object
 - toString
 - equals
 - 4 Zusammenfassung
 - Lessons learned
 - Ausblick
 - Fragen

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)

A large, light gray circular graphic containing a white number '4' followed by a white exclamation mark '!', positioned in the lower right quadrant of the slide.

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

```
65 class MateDrink extends SoftDrink {  
70     public String toString() {  
71         return super.toString() + " (of powerfull energy)";  
72     }  
73 }
```

toString - Praxis

Listing 20: src/SoftDrinks.java

```
7  MateDrink mate = new MateDrink();  
10 ColaDrink cola = new ColaDrink();  
  
15  
16 // output two objects  
17 System.out.println( mate );  
18 System.out.println( cola );
```

toString - Praxis

Listing 21: src/SoftDrinks.java

```
7  MateDrink mate = new MateDrink();  
10 ColaDrink cola = new ColaDrink();  
  
15  
16 // output two objects  
17 System.out.println( mate );  
18 System.out.println( cola );
```

Ausgabe

```
Softdrink named Default Drink, holding 0.01.  
    (of powerfull energy)  
Softdrink named Default Drink, holding 0.01.
```

Vergleich zwischen Mate und Cola



4!

Vergleich zwischen Mate und Cola

Listing 23: src/SoftDrinks.java

```
21 if( (Object)mate == (Object)cola ) {  
22     System.out.println(" Mate and Cola are the same!");  
23 } else {  
24     System.out.println(" Indeed, Mate and Cola are different!");  
25 }
```

Vergleich zwischen Mate und Cola

Listing 24: src/SoftDrinks.java

```
21  if( (Object)mate == (Object)cola ) {  
22      System.out.println(" Mate and Cola are the same!");  
23  } else {  
24      System.out.println(" Indeed, Mate and Cola are different!");  
25  }
```

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?



4!

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

```
28 MateDrink yourMate = new MateDrink();  
29 if( mate == yourMate ) {  
30     System.out.println("Your and mine mate are the same!");  
31 } else {  
32     System.out.println("Arrr, you have an other mate than me.");  
33 }  
34
```

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

```
28 MateDrink yourMate = new MateDrink();
29 if( mate == yourMate ) {
30     System.out.println("Your and mine mate are the same!");
31 } else {
32     System.out.println("Arrr, you have an other mate than me.");
33 }
34
```

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

```
13 public boolean equals(Object other) {  
14     if( other instanceof SoftDrink ) {  
15         SoftDrink otherDrink = (SoftDrink)other;  
16         return otherDrink.getName().equals(this.getName()) &&  
17             otherDrink.getVolume() == this.getVolume();  
18     } else {  
19         return false;  
20     }  
21 }  
22 }
```

equals-Methode

Listing 29: src/SoftDrinks.java

```
13 public boolean equals(Object other) {  
14     if( other instanceof SoftDrink ) {  
15         SoftDrink otherDrink = (SoftDrink)other;  
16         return otherDrink.getName().equals(this.getName()) &&  
17             otherDrink.getVolume() == this.getVolume();  
18     } else {  
19         return false;  
20     }  
21 }  
22 }
```

- 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

```
36 if( mate.equals(yourMate) == true ) {  
37     System.out.println(" Our mate soft drinks are the same");  
38 } else {  
39     System.out.println(" We do have different mate drinks!");  
40 }
```

Wertevergleich zwischen meiner und deiner Mate


Listing 31: src/SoftDrinks.java

```
36 if( mate.equals(yourMate) == true ) {  
37     System.out.println(" Our mate soft drinks are the same");  
38 } else {  
39     System.out.println(" We do have different mate drinks!");  
40 }
```

Ausgabe

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

Inhaltsverzeichnis

- 
- 1 Wiederholung
 - 2 Vererbung
 - Getränke
 - Klassenhierarchie
 - Zugriff auf Basisklasse
 - 3 Standard-Methoden
 - Object
 - toString
 - equals
 - 4 Zusammenfassung
 - Lessons learned
 - Ausblick
 - Fragen

Beispiel

Was passiert bei folgenden Programmfragmenten?

Listing 32: src/Human.java

```
4 public class Human{  
5     private String name;  
6     private int age;  
7  
8     public Human(){  
9         this.name = "";  
10        this.age = 0;  
11    }  
92 }
```

Listing 33: src/Human.java

```
53 public static void main(String[] args){  
54     // create moe  
55     Human moe = new Human();  
56     moe.age = -12;  
91 }
```

Beispiel

Was passiert bei folgenden Programmfragmenten?

Listing 34: src/Human.java

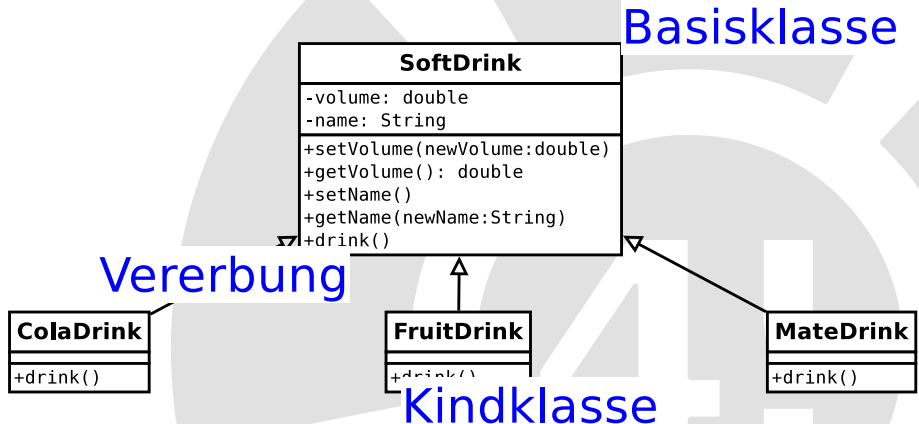
```
4 public class Human{  
5     private String name;  
6     private int age;  
7  
8     public Human(){  
9         this.name = "";  
10        this.age = 0;  
11    }  
92 }
```

Listing 35: src/Human.java

```
53 public static void main(String[] args){  
54     // create moe  
55     Human moe = new Human();  
56     moe.age = -12;  
91 }
```

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

Klassenhierarchie



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/>

???

4!