




EINFÜHRUNG IN DIE OBJEKTORIENTIERUNG

Von der Imperativen zur Objektorientierten
Programmierung



Einführung in die Objektorientierung

- Klassen vs. Objekte
- Konstruktoren
- Datenkapselung
 - *Sichtbarkeitsmodifikatoren*
 - *Getter/Setter Methoden*

KLASSEN VS. OBJEKTE



Klassen vs. Objekte

- **Klasse** beschreibt die Eigenschaften und Fähigkeiten gleichartiger Objekte
- Beispiel: Bauernhof Simulator
 - *Was ist ein Tier – explizit ein Schwein (Klasse oder Objekt?)*
 - Welchen Namen hat das Schwein?
 - Welches Gewicht hat das Schwein?
 - Wie Alt ist das Schwein?
 - Wo befindet es sich gerade?
 - *Die Antworten können nur auf ein bestimmtes Schwein gegeben werden. Somit bezieht es sich auf das Objekt (die Instanz) Schwein.*
 - *„Schwein“ bezieht sich auf die Klasse Schwein.*

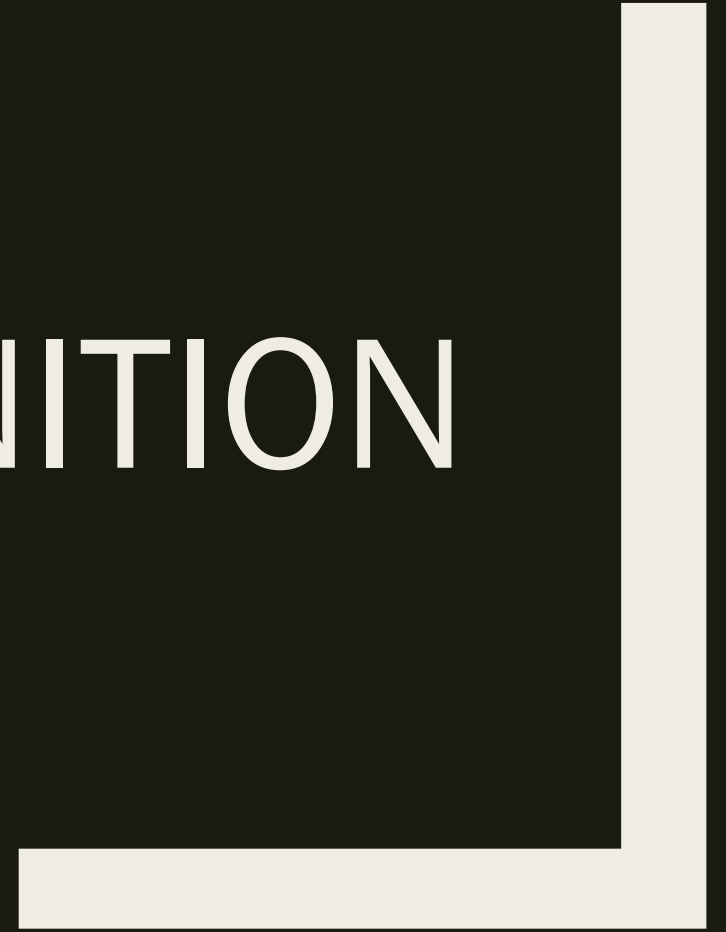
Klassen vs. Objekte

- Objekt ist individuelles Exemplar (physisch oder konzeptionell) mit bestimmten
 - *Eigenschaften (Attribute / Datenfelder)*
 - *Fähigkeiten (Methoden)* Beispiel: Bauernhof Simulator
- Beispiel: Das Schwein namens Rudy im Stall.
 - *Dieses Schwein ist eine Instanz der Klasse Schwein*
 - Welchen Namen hat das Schwein? Rudy.
 - Welches Gewicht hat das Schwein? 300 Kilo
 - Wie Alt ist das Schwein? 4 Jahre
 - Wo befindet es sich gerade? im Stall
- Mit den Antworten beschreiben wir den „Zustand“ eines Speziellen Objektes.

Klassen vs. Objekte

- Objekt ist eine Instanz, das heißt eine konkrete Umsetzung einer Klasse.
- Eine Klasse bildet eine gedankliche oder reale Einheit.
- Wozu Objektorientierung
 - *Programmierung: Abbild der Realen Welt*
 - *Einfache Wiederverwendbarkeit und Wartung*

KLASSENDERFINITION



Klassendefinition

Allgemeiner Aufbau einer Klasse

```
/* Dokumentation */
```

```
public class <name> {
```

Datenfelder

Konstruktoren

Methoden

```
}
```

Dokumentation:

```
/* mehrzeilige Kommentare */
```

```
// einzeilige Kommentare
```

public class <name>: Kopf / Signatur der Klasse

public und **class**: Schlüsselwörter oder reservierte Wörter; (key words)

Rumpf / Body der Klasse zwischen { ... }

< >: Platzhalter für einen Bezeichner
(Name des Datenelementes) oder Identifikator

OBJEKTE ERZEUGEN



Objekte erzeugen

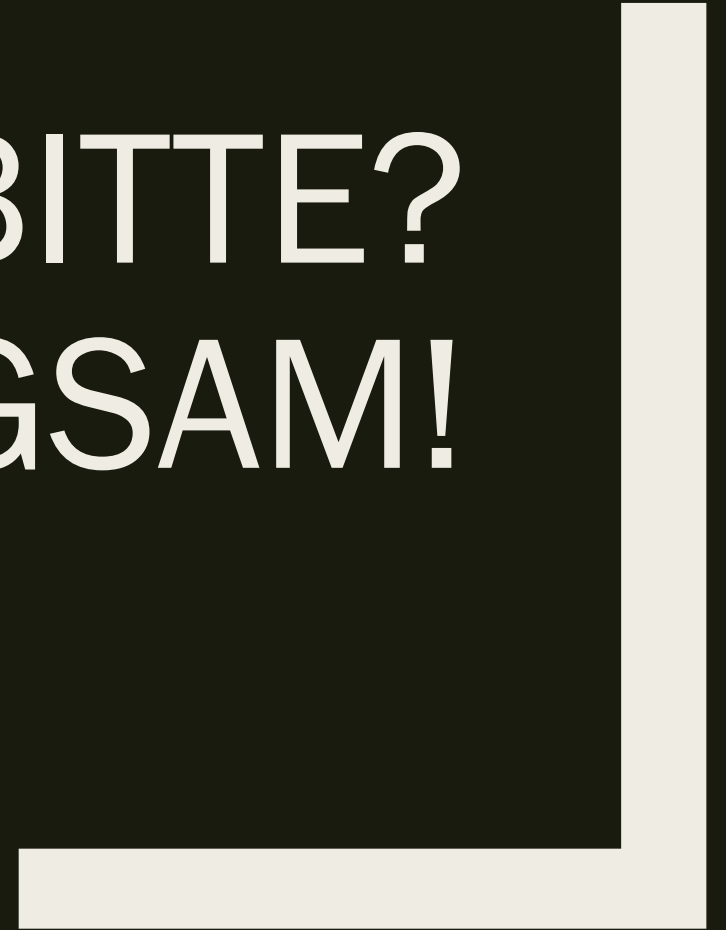
- Objekt erzeugen mit Java Code
- in 2 Schritten.

```
Circle sun;  
...  
...  
sun = new Circle();
```

- in einem Schritt

```
Circle sun = new Circle();
```

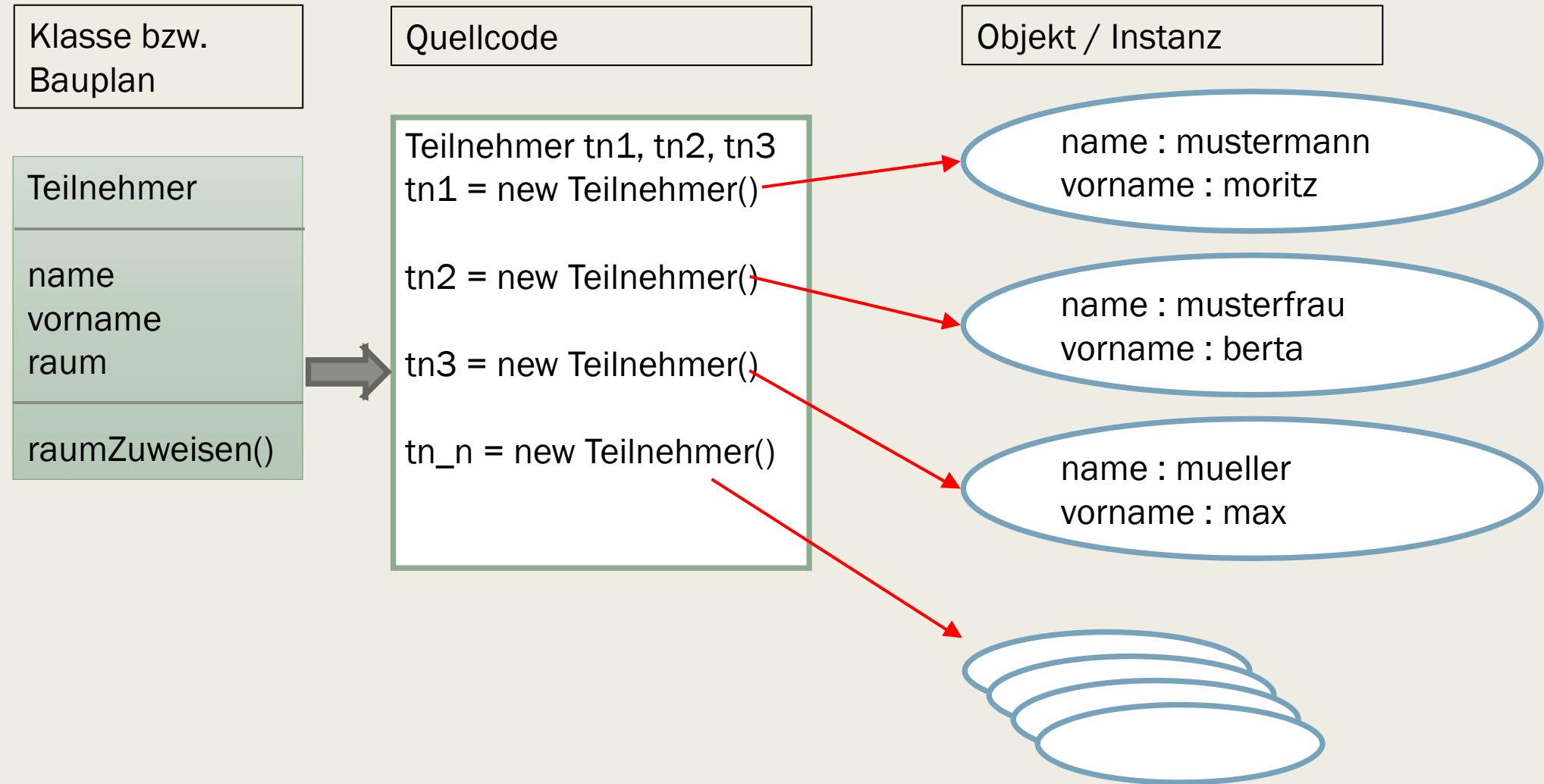
WIE BITTE?
NOCHMAL LANGSAM!



Erläuterung

- Eine Klasse definiert den Bauplan, abstraktes Schema
- Ein Objekt ist konkretes Exemplar / Realisierung dieses Schemas
- Objekt einer Klasse muss explizit erzeugt werden
- Von einer Klasse beliebig viele Objekte ableitbar
- **new** <Klassenname>()
Erzeugt ein einzelnes neues Objekt auf Grundlage einer Klasse

Erläuterung



new <Klassenname>()

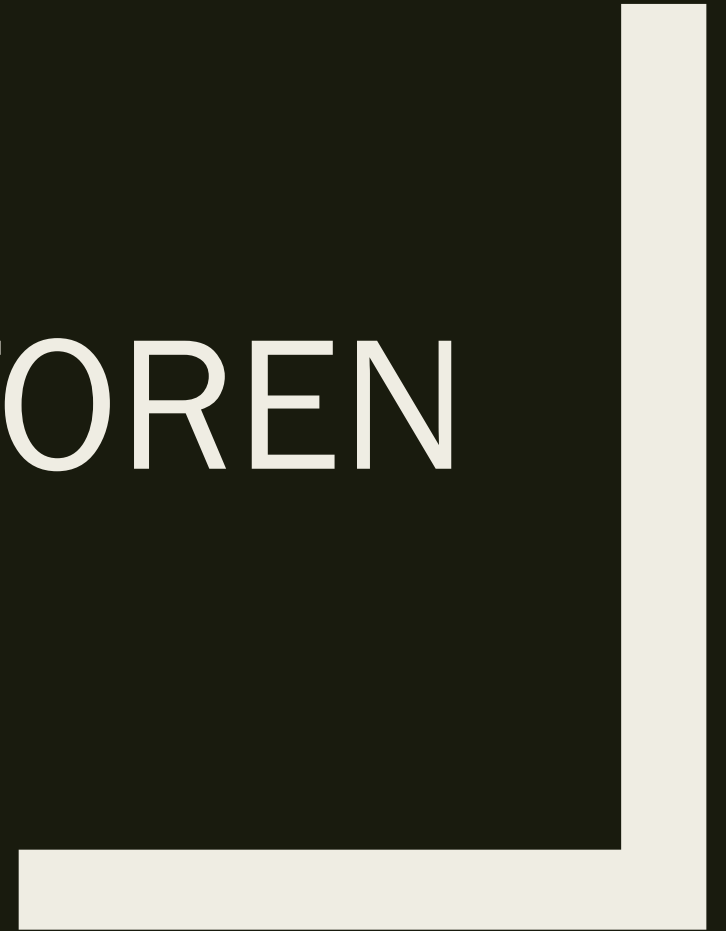
- Erzeugt ein neues Objekt auf Grundlage einer Klasse.
 - *Allokieren des dafür erforderlichen Speicherplatzes*
 - *Initialisieren der Datenfelder*
- Mehrere Objekte bekommen durch mehrfaches aufrufen von „new“
- Der wert des new-Ausdrucks ist die Referenz auf das neue Objekt.

tn1 → Referenz auf ein Objekt

Zeigt auf das neue Objekt im Speicher

Im Quellcode können wir das Objekt nur über seine Referenz ansprechen

KONSTRUKTOREN



Konstruktoren

- Jede Klasse hat einen oder mehrere Konstruktoren für ihre Objekte.
- Konstruktoren stellen eine Art Spezieller Methoden dar.
 - *Ein Konstruktor:*
 - heißt exakt wie die Klasse
 - hat keinen Rückgabewert, auch nicht void
 - Kann Parameter für die Initialisierung des Zustand des neuen Objekts haben.

Konstrukturen

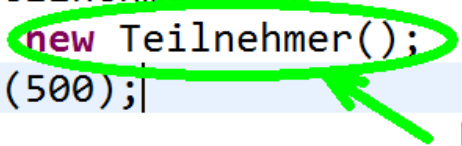
```
2 public class Teilnehmer {  
3  
4     public String name;  
5     public String vorname;  
6     public int raum;  
7  
8     public static int anzahlTeilnehmer = 0;  
9  
10    public Teilnehmer() {  
11        name = "max";  
12        vorname = "mustermann";  
13        raum = 1;  
14    }  
15  
16    public void raumZuweisen(int raumnummer) {  
17        this.raum = raumnummer;  
18    }  
19  
20    public static void ermittleAnzahlTeilnehmer() {  
21        System.out.println("Anzahlteilnehmer = " + anzahlTeilnehmer);  
22    }  
23 }
```

Gleiche Benennung wie die Klasse

Konstruktor

- Ein Konstruktor ist keine Methode eines Objektes, sondern ein Weg Objekte zu erzeugen.
- Ein anderes Objekt kann den Konstruktor verwenden, um sich Objekte dieser Klasse zu erzeugen und zu nutzen.

```
1  
2 public class Verwaltung {  
3  
4     public void erzeugeTeilnehmer() {  
5         Teilnehmer tn1 = new Teilnehmer();  
6         tn1.raumZuweisen(500);  
7     }  
8 }  
9
```



Hier entsteht ein neues Objekt

Konstrukturen

Default Konstruktor

```
public Teilnehmer() {  
    name = "max";  
    vorname = "mustermann";  
    raum = 1;  
}
```

Voll-Parametrisierter Konstruktor

```
public Teilnehmer(String name, String vorname, int raum) {  
    this.name = name;  
    this.vorname = vorname;  
    this.raum = raum;  
}
```

Default Konstruktoren

- Der default Konstruktor (Eigenname) ist ein Konstruktor, dem keine Parameter übergeben werden. Dieser Konstruktor dient dazu, Objekte mit „default“ (Standard) werten zu versehen.

```
public Teilnehmer() {  
    name = "max";  
    vorname = "mustermann";  
    raum = 1;  
}
```

Parametrisierte Konstruktoren

- Die Parametrisierten Konstruktoren (Eigenname) dienen dazu, um Objekte mit den Übergebenen Werten zu befüllen. Konstruktoren können alle Attribute als Parameter bereitstellen müssen es jedoch nicht.
- Beispiel Voll Parametrisierter Konstruktor

```
public Teilnehmer(String name, String vorname, int raum) {  
    this.name = name;  
    this.vorname = vorname;  
    this.raum = raum;  
}
```

Kopier-Konstruktor

- Mit einem Kopier-Konstruktor können Kopien eines Objektes erstellt werden. Dazu werden die Werte des Originals auf die eigene Instanz übertragen.

```
public Teilnehmer(Teilnehmer original) {  
    this.name = original.name;  
    this.vorname = original.vorname;  
    this.raum = original.raum;  
}
```

SICHTBARKEITSMODIFIKATOREN



Sichtbarkeitsmodifikatoren

- In Java stehen uns 4 Sichtbarkeitsmodifikatoren zur Verfügung.
 - Öffentlich - *public*
 - Geschützt - *protected*
 - Paketsichtbar - *default*
 - Privat – *private*
- Bis auf Paketsichtbar hat jeder Modifikator ein Schlüsselwort.
- Die Sichtbarkeitsmodifikatoren können auf Klassen, Attribute und Methoden angewendet werden.

Öffentlich - public

- Als public deklarierte Elemente können von allen anderen Klassen genutzt und drauf zugegriffen werden.
- Attribute die public sind, können von jeder anderen Klassen gelesen und verändert werden.

- Beispiel:

```
public String name;
```

```
public void offen(){  
}
```

Geschützt - protected

- Als protected deklarierte Elemente können nur von Klassen im selben Paket oder von Kindklassen (Subclasses) genutzt werden.
- Attribute die protected sind, können von Klassen im selben Paket wie auch von Kindklassen gelesen und verändert werden.

- Beispiel:

```
protected String name;
```

```
protected void geschuetzt(){  
}
```

Paketsichtbar – default

- Als default deklarierte Elemente können nur von Klassen im selben Paket genutzt werden.
- Attribute die default sind, können von Klassen im selben Paket gelesen und verändert werden.

- Beispiel:

```
String name;
```

```
void paketsichtbar(){  
}
```

Privat – private

- Als private deklarierte Elemente können nur von der Klassen genutzt werden, wo diese Elemente erstellt wurden. Andere Klassen haben keinen Zugriff auf diese Elemente
- Attribute die private sind, können von keiner anderen Klasse gesehen und verändert werden. Nur die Klasse, wo diese Attribute angelegt wurden, hat darauf zugriff.

- Beispiel:

```
private String name;
```

```
private void nurFuerDieKlasseSichtbar(){  
}
```

GETTER / SETTER



Getter / Setter

- In der OOP gibt es den Ansatz der Datenkapselung.
- Dies bedeutet auf die Attribute einer Klasse, darf nur die Klasse selbst direkt zugreifen.
- Andere Außenstehende Klassen und damit auch Objekte, müssen sich der entsprechenden Getter / Setter Methoden bedienen.
- Durch das Nutzen von Getter und Setter Methoden entsteht die Möglichkeit, nur vom Design der Klasse vorgesehen Werte in die Attribute schreiben zu lassen.
- Konsequenterweise müssen dann alle Attribute einer Klasse mit den Sichtbarkeitsmodifikator „private“ versehen werden.

Getter / Setter

■ Getter Methoden

- *get wie holen, wird dem Attributnamen vorangestellt.*
- *Liest den wert eines Attributes und gibt diesen zurück*

Aufbau: Beispiel eines int Attributes

```
public int get<Attributname>() {  
    return this.attributname;  
}
```

■ Setter Methoden

- *set wie festlegen, wird dem Attributnamen vorangestellt.*
- *Schreibt einen wert in das Attribut und liefert nichts zurück.*

Aufbau: Beispiel eines int Attributes

```
public void set<Attributname>(int attr) {  
    this.attributname = attr;  
}
```

TOSTRING

Visitenkarten Methode



public String toString()

- Er gehört zum „guten“ Ton in Java eine toString() Methode zu schreiben.
- Diese Methode sollte die Wichtigsten Elemente ihrer Klasse zusammenfassen und als String zurückliefern.
- Diese Methode ist implizit in allen Klassen vorhanden, da „Object“ eine toString methode besitzt und alle Klassen ohne Ausnahme von Object abgeleitet sind.
- Die von Object „geerbte“ toString() Methode liefert jedoch nur einen Pseudo Adressbereich, der relativ wenig über den Inhalt einer Klasse ausgibt.

GRUNDGERÜST EINER KLASSE



Grundgerüst einer Klasse

```
public class Teilnehmer{  
    // Attribute  
    private String name;  
    // Konstruktoren  
    public Teilnehmer(){  
        }  
    // Getter Setter  
    public String getName(){  
        }  
    public void setName(String name){  
        }  
    // toString Methode  
    public String toString(){  
        }  
}
```