

# Lektion 9

Klassen  
Konstruktoren  
(Objekt-/Instanz-)Methoden

# Klassen

Getter und Setter  
Methoden  
Modifier

# Klassen

- Jede Klasse befindet sich in einer eigenen Datei!
- Der Klassenname beginnt immer mit einem **Großbuchstaben**.
- Eine Klasse hat Eigenschaften: Attribute/Felder und Methoden

Klassenname	Punkt
Attribute	x- und y-Koordinate
Methoden	Punkt verschieben

```
public class Punkt
{
    //Attribute/Felder beschreiben den Zustand
    int x;
    int y;

    //Methoden beschreiben das Verhalten der Objekte
    public void verschiebePunkt(int zielX, int zielY) {
        x = zielX;
        y = zielY;
    }
}
```

Über den Punktoperator . kann auch auf die  
**Methoden**  
des Objekts zugegriffen werden.

Allgemein:

**<Name der Referenz>.<Methodenname>(<Arg1>, <Arg2>, ...)**

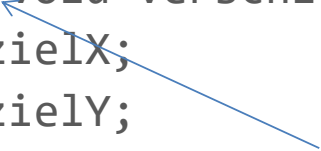
Beispiel:

```
Punkt punkt = new Punkt();  
punkt.verschiebePunkt(5,7)
```

Methoden und Attribute werden standardmäßig dem Objekt zugeordnet,  
es sei denn der Modifier **static** wird verwendet.

```
public class Punkt
{
    //Attribute/Felder beschreiben den Zustand
    int x;
    int y;

    //Methoden beschreiben das Verhalten der Objekte
    public void verschiebePunkt(int zielX, int zielY) {
        x = zielX;
        y = zielY;
    }
}
```



kein static

In objektorientierten Programmiersprachen erfolgt der Zugriff auf den **Zustand** eines Objekts i.d.R. über dessen Methoden.



```
public class Person {
```

```
    String vorname;
```

```
    ...
```

```
    public String getVorname() {
```

```
        return vorname;
```

```
    }
```

```
    public void setVorname(String v) {
```

```
        vorname = v;
```

```
    }
```

```
    ...
```

```
}
```

getter-Methode für „lesenden Zugriff“

setter-Methode für „schreibenden Zugriff“

Konvention: get + Attributname beginnend mit Großbuchstabe  
bzw. set + Attributname beginnend mit Großbuchstabe

Beispiel:

set + vorname ➤ set + Vorname ➤ setVorname  
get + vorname ➤ get + Vorname ➤ getVorname

Ausnahme beim Typ boolean:

**boolean** primzahl;

is + primzahl ➤ is + Primzahl ➤ isPrimzahl

Bisher konnten wir mit:

**joe.vorname**

einfach auf ein Attribut zugreifen.

Das lässt sich auch unterbinden durch  
den Modifier

**private**



# Modifier public vs. private

```
public class Person {  
    private String vorname;  
    public String nachname;  
  
    public String getVorname() {  
        return vorname;  
    }  
    public void setVorname(String v) {  
        vorname = v;  
    }  
}
```

```
public static void main (String[] args)  
{  
    Person p1 = new Person();  
    p1.setVorname("joe");  
    p1.nachname = "Cool";  
    System.out.println(p1.getVorname());  
    System.out.println(p1.nachname);  
}
```

- Modifier **public** bedeutet, dass auf dieses Attribut von überall im Programm zugegriffen werden darf.
- Modifier **private** bedeutet, dass auf dieses Attribut nur von Methoden des Objekts/der Klasse selbst zugegriffen werden kann, im Bsp durch *setVorname* und *getVorname*.

## Modifier private – Zugriff aus eigener Klasse

```
public class Person {  
    private String vorname;  
    public String nachname;  
    public String getVorname() {  
        return vorname;  
    }  
    public void setVorname(String v) {  
        vorname = v;  
    }  
  
    public static void main(String[] args)  
    {  
        Person p = new Person();  
        p.vorname = "Joe";  
    }  
}
```

Zugriff ist erlaubt. Er erfolgt aus  
der selben Klasse.

# Modifier private – Zugriff aus anderer Klasse

```
public class Person {  
    private String vorname;  
    public String nachname;  
    public String getVorname() {  
        return vorname;  
    }  
    public void setVorname(String v) {  
        vorname = v;  
    }  
}  
public class Main {  
    public static void main(String[] args)  
    {  
        Person p = new Person();  
        p.vorname = "Joe";  
    }  
}
```

Zugriff ist nicht erlaubt.  
Er erfolgt aus einer anderen Klasse.  
Compiler-Fehler:  
The field Person.vorname is not visible

## Modifier private – Zugriff aus anderer Klasse

```
public class Person {  
    private String vorname;  
    public String nachname;  
    public String getVorname() {  
        return vorname;  
    }  
    public void setVorname(String v) {  
        vorname = v;  
    }  
}  
public class Main {  
    public static void main(String[] args)  
    {  
        Person p = new Person();  
        p.setVorname("Joe");  
    }  
}
```

Zugriff über die Methode ist erlaubt.  
Die Methode ist öffentlich.

# Modifier public – Zugriff von überall möglich

```
public class Person {  
    private String vorname;  
    public String nachname;  
    public String getVorname() {  
        return vorname;  
    }  
    public void setVorname(String v) {  
        vorname = v;  
    }  
}  
public class Main {  
    public static void main(String[] args)  
    {  
        Person p = new Person();  
        p.nachname = "Cool";  
    }  
}
```

Zugriff auf das Attribut ist erlaubt.  
Es ist öffentlich.

# Musterbeispiel: Zugriff auf Objekte über Methoden

```
public class Person
{
    //Attribute
    private String vorname;
    private String nachname;
    private String username;

    //Methoden
    public String getVorname() {
        return vorname;
    }
    public void setVorname(String v) {
        vorname = v;
    }
    public String getNachname() {
        return nachname;
    }
    public void setNachname(String n) {
        nachname = n;
    }
}
```

```
    public String getUsername() {
        return username;
    }
    public void setUsername(String u) {
        username = u;
    }
}

public class Main
{
    public static void main(String[] ar)
    {
        Person joe;
        joe = new Person();
        joe.setNachname("Cool");
        joe.setVorname("Joe");
        joe.setUsername("jcool");
        System.out.println(joe.getVorname()
            + " " + joe.getNachname());
    }
}
```

# Konstrukturen

```
Person joe = new Person();  
joe.setVorname("Joe");  
joe.setNachname("Cool");  
joe.setUsername("jcool");
```

Wenn man viele Objekte erstellt und direkt mit Daten belegen muss, ist es umständlich jedes Attribut durch einen Setter zu belegen.



Es ist möglich, diese Daten direkt bei der Erstellung mitzugeben.



```
Person joe  
  = new Person("Joe", "Cool", "jcool");
```



Damit die Übergabe funktioniert, muss die Person-Klasse angepasst werden.

Z. B. bei der Erstellung des Scanners geben wir mit, dass er von System.in lesen soll:

```
Scanner sc = new Scanner(System.in);
```



# Konstruktoren

- Wenn ein Objekt durch den Aufruf des new-Operators erstellt wird, wird dessen Konstruktor aufgerufen.
- Ein Konstruktor ist wie eine Methode, hat aber keinen Rückgabotyp.
- Ein Konstruktor ohne Parameter nennt sich **Standardkonstruktor**.

```
public class Person
```

```
{
```

```
    public Person() ← Standardkonstruktor hat keine Parameter.
```

```
{
```

Hier können Anweisungen platziert werden,

```
}
```

← die beim Erzeugen des Objekts ausgeführt werden sollen,

z. B. eine Konfiguration aus einer Datei einlesen, etc.

```
}
```

Der Name des Konstruktors entspricht  
dem Namen der Klasse

- Wenn kein Konstruktor im Quellcode auftaucht, wird dennoch der Standardkonstruktor bei der Kompilierung „ergänzt“.

# Konstrukturen

- Einem Konstruktor können **Argumente übergeben werden**, mit denen das Objekt erstellt werden soll.

```
public class Person
```

```
{  
    String vorname;  
    String nachname;  
    String username;
```

Wird ein **eigener Konstruktor** definiert, wird **kein** Standardkonstruktor automatisch „ergänzt“. Er kann bei Bedarf separat definiert werden.


```
    public Person(String v, String n, String u)  
    {  
        vorname = v;  
        nachname = n;  
        username = u;  
    }  
}
```

```
public class Main  
{  
    public static void main (String[] args)  
    {  
        Person p;  
        p = new Person("Joe", "Cool", "jcool");  
    }  
}
```

# this-Referenz

- **this** ist eine Referenz auf das aktuelle Objekt.
- Über **this** kann
  - in jeder Objektmethode (Nicht-Static-Methode)
  - in jedem Konstruktorauf das aktuelle Objekt und dessen Attribute zugegriffen werden.

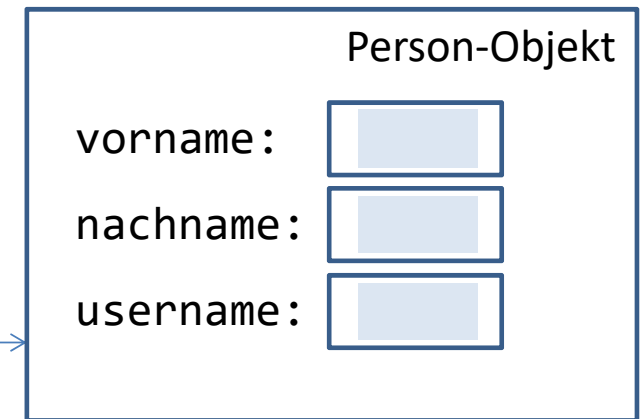
```
public class Person {  
    String vorname;  
    String nachname;  
  
    public Person(String vorname, String nachname)  
    {  
        this.vorname = vorname;  
        this.nachname = nachname;  
    }  
}
```



# this-Beispiel

```
public class Person {  
    String vorname;  
    String nachname;  
  
    public Person(String vorname, String nachname) {  
        this.vorname = vorname;  
        this.nachname = nachname;  
    }  
  
    public static void main(String[] args) {  
        Person joe = new Person("Joe", "Cool");  
    }  
}
```

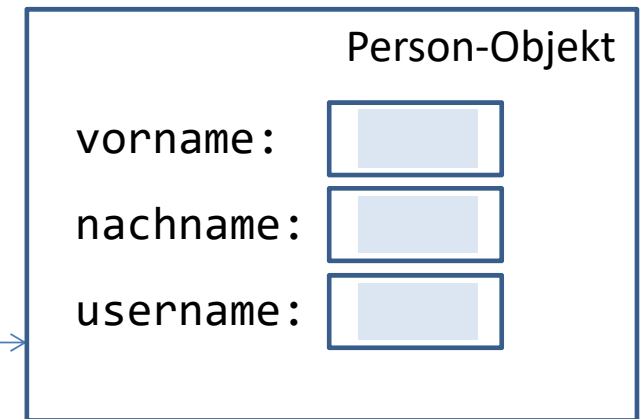
joe →



# this-Beispiel

```
public class Person {  
    String vorname;  
    String nachname;  
  
    public Person(String vorname, String nachname) {  
        this.vorname = "Joe"  
        this.nachname = "Cool"  
    }  
  
    public static void main(String[] args) {  
        Person joe = new Person("Joe","Cool");  
    }  
}
```

joe →



# Gesamtbeispiel: Person



```
public class Person
{
    private String vorname;
    private String nachname;
    private String username;

    public Person() {
    }

    public Person(String vorname,
        String nachname, String username)
    {
        this.vorname = vorname;
        this.nachname = nachname;
        this.username = username;
    }
}
```


```
    public String getVorname() {
        return vorname;
    }
    public void setVorname(String vorname) {
        this.vorname = vorname;
    }
    public String getNachname() {
        return nachname;
    }
    public void setNachname(String nachname) {
        this.nachname = nachname;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
}
```

Um die Komplexität und Kopplung in Programmen zu verringern, kann der Zugriff auf Variablen über die Methoden eingeschränkt werden.

Soll bspw. auf den Vornamen von außerhalb der Klasse nur lesend zugegriffen werden können, wird kein setter zur Verfügung gestellt:

```
public class Person {  
    String vorname;  
    ...  
    public String getVorname() {  
        return vorname;  
    }  
    ...  
}
```

getter-Methode für „lesenden Zugriff“



```
public class Punkt {  
    int x;  
    int y;
```

```
    public void verschiebePunkt(int zielX, int zielY) {  
        x = zielX;  
        y = zielY;  
    }  
}
```

Bei der Klasse Punkt werden überhaupt keine setter benötigt, da die x- und y-Koordinaten nur gemeinsam verändert werden können.

Die Methode *verschiebePunkt* hat einen treffenden Namen erhalten, der möglichst genau das von der Methode erwartete Verhalten wiedergibt.

Dadurch sieht der Programmierer nur eine kleine Anzahl aussagekräftiger Methoden.

Das Prinzip nur die Details nach außen zu geben, die unbedingt notwendig sind, heißt **Datenkapselungsprinzip (Information Hiding)**.



```
public class Pizza {  
    String name;  
    int durchmesser;  
    float preis;
```

## Zurück zum Pizzabeispiel:

```
    public Pizza(String name, int durchmesser, float preis) {  
        this.name = name;  
        this.durchmesser = durchmesser;  
        this.preis = preis;  
    }
```

Konstruktor

```
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public int getDurchmesser() {  
        return durchmesser;  
    }  
    public void setDurchmesser(int durchmesser) {  
        this.durchmesser = durchmesser;  
    }  
    public float getPreis() {  
        return preis;  
    }  
    public void setPreis(float preis) {  
        this.preis = preis;  
    }  
}
```

getter und setter

```
public class Speisekarte
{
    public static void main(String[] args)
    {
        final int ANZAHL_PIZZEN = 15;
        Pizza[] pizzas = new Pizza[ANZAHL_PIZZEN];
        ...
    }
}
```

Wir haben mit

```
Pizza[] pizzas = new Pizza[ANZAHL_PIZZEN];
```

15 Referenzen auf Pizza-Objekte  
erstellt, aber noch keinen Speicher  
für die Objekte reserviert.

Jedes Array ist übrigens auch  
ein Objekt mit dem Attribut  
**length**.

```
final int ANZAHL_PIZZEN = 15;  
Pizza[] pizzas = new Pizza[ANZAHL_PIZZEN];
```

`pizzas.length` wird zu 15 ausgewertet!

```
public class Speisekarte
{
    public static void main(String[] args)
    {
        final int ANZAHL_PIZZEN = 15;
        Pizza[] pizzas = new Pizza[ANZAHL_PIZZEN];
        pizzas[0] = new Pizza("Pizza Margherita", 26, 4.5f);
        pizzas[1] = new Pizza("Pizza Peperoniwurst", 26, 5.5f);
        pizzas[2] = new Pizza("Pizza Schinken", 26, 5.5f);
        pizzas[3] = new Pizza("Pizza Speciale", 26, 6.5f);
        pizzas[4] = new Pizza("Pizza Vegetaria", 26, 6f);
        ...
    }
}
```

Jede der 15 Referenzen verweist jetzt  
auf ein neu erstelltes Pizza-Objekt!

## Wie geben wir die Speisekarte aus?

5 Leerzeichen zwischen  
längstem Pizzanamen und  
Überschrift *Durchmesser*

5 Leerzeichen

Name	Durchmesser	Preis in EUR
Pizza Margherita	26cm	4.50
Pizza Peperoniwurst	26cm	5.50
Pizza Schinken	26cm	5.50
Pizza Speciale	26cm	6.50
Pizza Vegetaria	26cm	6.00
...		

Wir müssen wissen, wie lange der  
längste Pizzaname ist!

Wir müssen wissen, wie lange der  
längste Pizzaname ist!

```
int laengsterName = 0;
for (int i = 0; i < pizzas.length; i++)
{
    int laengeAktuellerPizza = pizzas[i].getName().length();
    if (laengsterName < laengeAktuellerPizza)
        laengsterName = laengeAktuellerPizza;
}
```

Durchmesser und Preis sind einfacher:

```
int durchmesserLaenge = "Durchmesser".length();
int preisLaenge = "Preis in EUR".length();
```

## Ausgabe der Speisekarte

```
for (int i = 0; i < pizzas.length; i++)
{
    System.out.print(pizzas[i].getName());
    //5 = mind. Anzahl Leerzeichen
    int anzahlLeerzeichen = laengsterName - pizzas[i].getName().length()+5;
    System.out.printf("%" + anzahlLeerzeichen + "s", "");

    //-2, da "cm" angefügt wird
    anzahlLeerzeichen = durchmesserLaenge -2;
    System.out.printf("%" + anzahlLeerzeichen + "dcm",
        pizzas[i].getDurchmesser());

    //5 = Abstand zwischen Durchmesser und Preis
    anzahlLeerzeichen = preisLaenge +5;
    System.out.printf("%" + anzahlLeerzeichen + ".2f",
        pizzas[i].getPreis());
    System.out.println();
}
```

printf-Legende:  
%s für String  
%d für int  
%f für float  
%5f float rechtsbündig  
auf 5 (Vorkomma-)Stellen  
ausgeben