UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Programmierkurs Java

Objektorientierte Programmierung

Raphael Allner
Institut für Telematik
05. November 2019

Überblick

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

1. Objektorientierung
   - Objekte und Klassen
   - Verhalten und Zustand: Attribute und Methoden
2. Objektorientierung in Java
   - Deklaration und Verwendung von Klassen
   - Implementierung von Methoden: Attribute verwenden
3. Konstruktoren: Objekte initialisieren
4. Geheimnisprinzip
   - Schnittstellen und Interna
   - Syntax für die Sichtbarkeit von Eigenschaften
5. Klassenattribute und -methoden VS. Objektattribute und -methoden

Übungsaufgabe

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

1. Laden Sie die Dateien `Cat.java` und `Main.java` aus dem **Moodle** herunter
2. Die Aufgabenstellung entnehmen Sie Main.java
3. Verfolgen Sie die Vorlesung um die Aufgaben zu erfüllen
4. Überprüfen Sie Ihren Code:
    1. Kommandozeilentool starten und in das Verzeichnis der .java Dateien navigieren
    2. Kompilieren mit `javac Main.java`
    3. Ausführen in der JVM mit `java Main`
    4. **`Cat.java müssen Sie nicht extra kompilieren.`**

Objektorientierte Programmierung

Objektorientierung

Objektorientierung
Motivation

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Bisher:

- Alles in **einer Datei**
- **Methoden** strukturieren Code
- Alles statisch (**`static`**)
- Übergabe von Daten per **Parameter** und **Rückgabewerte**
- **Globale Variablen** für wichtige Daten

Einfach Programme zu schreiben ist möglich.

Herausforderung in großen und komplexen Programmen:

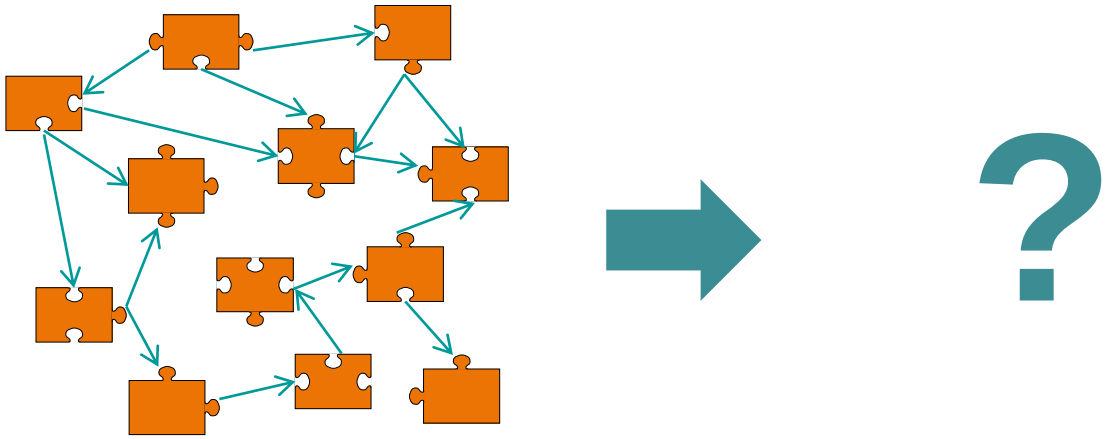
- In welcher **Reihenfolge** wird der Code ausgeführt?
- Welche **Datenstrukturen** werden wo und wie verwendet?
- Welcher **Code interagiert** mit welchem anderen Code?

Objektorientierung
Motivation

Notwendig:

- Kapselung und Aufteilung einzelner **Funktionseinheiten**
- **Gruppierung** zusammengehöriger **Daten** und **Operationen**
- **Zugriff von außen** auf diese **steuern**

Was könnten diese Funktionseinheiten aussehen?



Objektorientierung
Motivation

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Objekte der realen Welt werden im Code modelliert

Beispiel „elektronischer Busfahrplan“

- Fahrgäste
- Tarifzonen
- Busse
- Fahrer*in
- Haltestellen
- Usw.

Objektorientierung
Objekt - Definition

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Ein Objekt ist eine **Repräsentation** ...一个对象就是一个代表

- ... eines Gegenstandes oder Sachverhalts der realen Welt
- ... eines rein gedanklichen Konzepts

Ein Objekt ist gekennzeichnet durch:

- Eine **eindeutige Identität**, die es von anderen Objekten unterscheidet 一个与自己等价的对象
  - Dieser Bus / ein anderer Bus oder diese Katze / eine andere Katze
- Einen **Zustand**: repräsentiert durch Attribute 一个特征或是状态
  - `int[] gpsKoordinaten;`
  - `String katzenName;`
- Ein **Verhalten**: repräsentiert durch Methoden 一个行为
  - `fahrenVorwaerts();`
  - `katzeFuettern();`

Objektorientierung
Objekt - Definition

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Der Zustand eines Objekts:

- **Zu einem bestimmten Zeitpunkt** 一个对象的状态只用于当前时间点
- Entspricht der **Belegung der Attribute** des Objekts
  - zu diesem Zeitpunkt

Das Verhalten eines Objekts:

- Wird durch **Methoden** dargestellt 一个对象的行为由Methode来实现
- Entspricht einer **programmiersprachlichen Umsetzung** von **Prozeduren** bzw. **Funktionen**, denen Parameter übergeben werden können.
- Diese ermöglichen einen **gesteuerten Zugriff auf den Zustand** des Objektes

Objektorientierung
Varianten von Objektmodellen

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Identitätsbasiert

每个对象都有自己等价的另一个对象

- **Jedes Objekt** innerhalb eines Systems besitzt seine **eigene Identität**
- Zwei Objekte o_1 und o_2 sind gleich, wenn der Wert, der ihre Identität *(Speicheradresse)* bestimmt, gleich ist
- Der **Zustand** ist in diesem Fall **ohne Bedeutung**

状态在这种情况下没有意义

Java

Wertbasiert

- Es existieren **keine** speziellen **Objektidentitäten**
- **Einzigartigkeit** eines Objekts basiert ausschließlich auf seinem **Zustand**

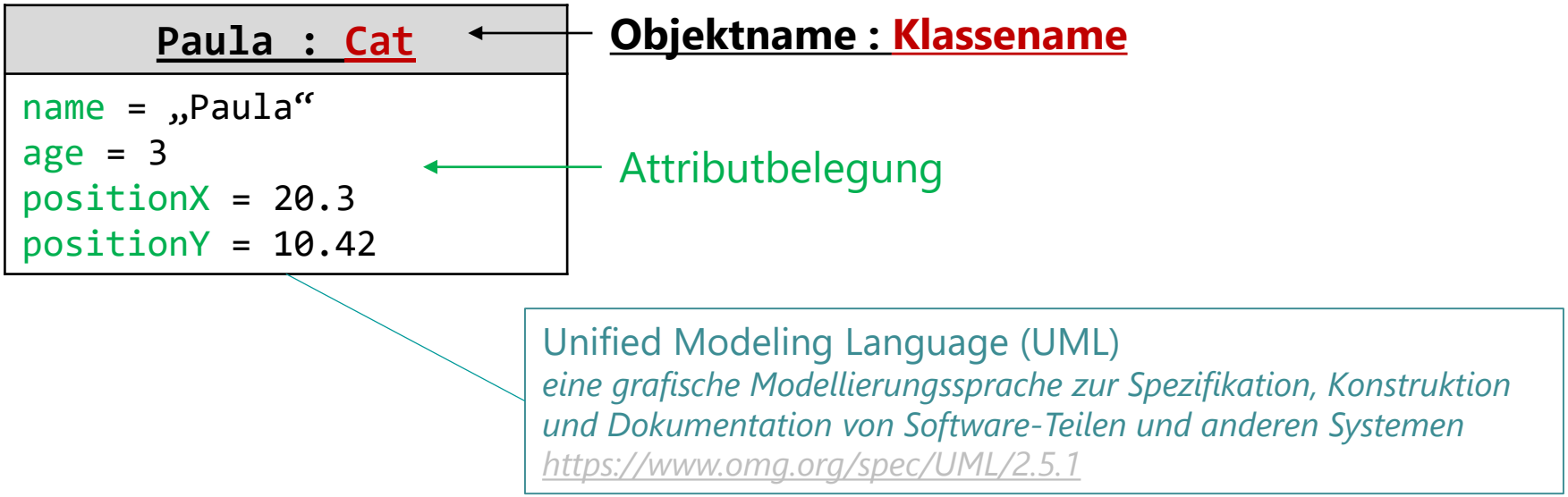
一个对象没有特定的等价对象
它基于特征的描述

Hybrid

- Der Modellierer oder **Programmierer legt fest**, wie die Einzigartigkeit eines Objektes bestimmt wird

Objektorientierung
Objekt – Objektdiagramm (UML)

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK



Muss jedes Objekt immer neu implementiert werden?

- Nein!
- Man fasst **ähnliche Objekte** zu einer **Klasse** zusammen

Objektorientierung
Klasse - Definition

Class是对这个对象的描述

Eine Klasse ist die Beschreibung/der Bauplan eines Objekts

Besteht aus einer Menge von:

- Attributen (Statische Eigenschaften) 特征
- Methoden (Verhalten) 行为
- Konstruktoren (Beschreibungen, wie neue Objekte dieser Klasse erzeugt / konstruiert werden können) 描述

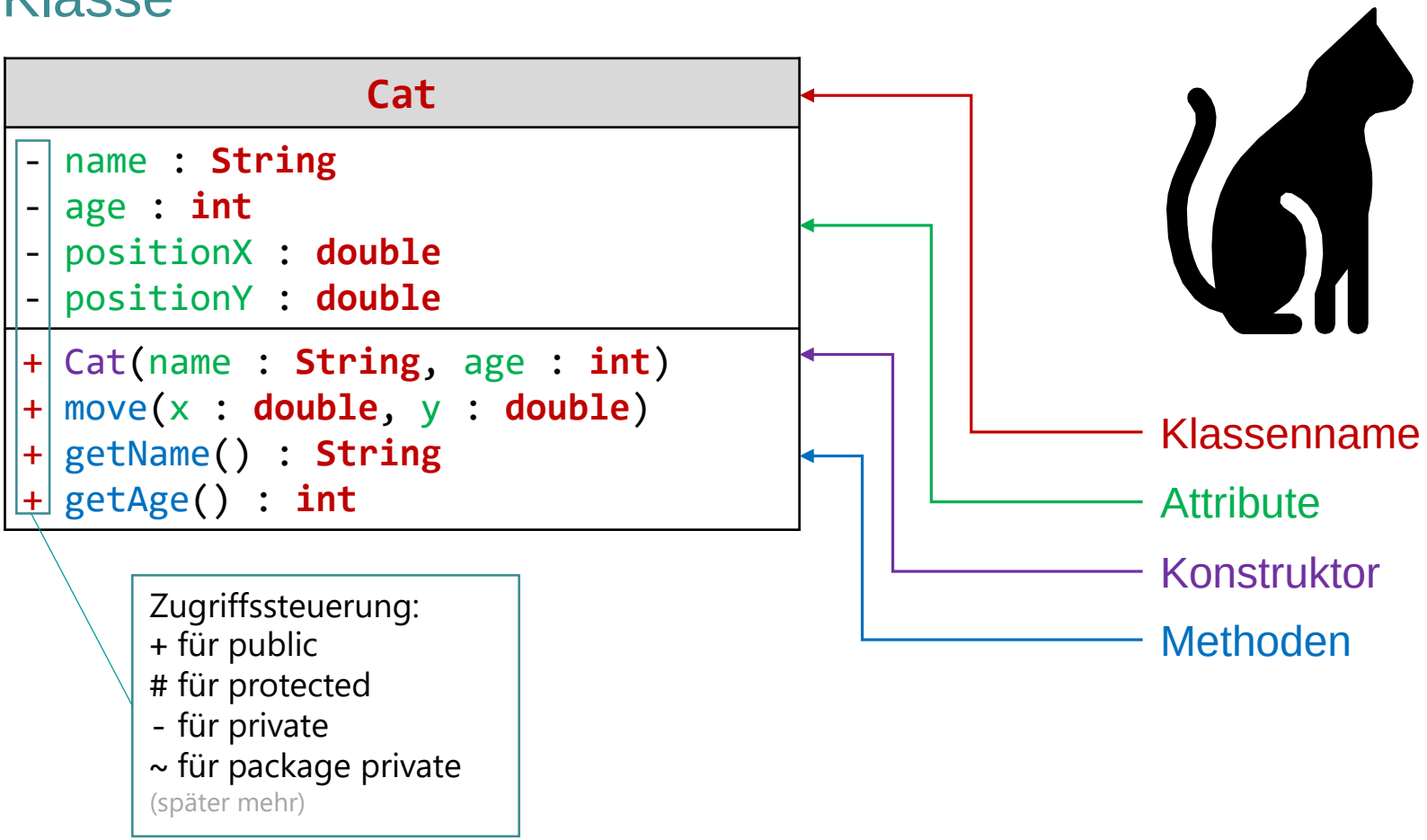
Sie definiert:

- Die **Interne Repräsentation der Daten** eines Objekts
- Das **durch Schnittstellen spezifizierte Verhalten** eines Objekts

Objektorientierung
Klasse - Klassendiagramm (UML)

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Klasse

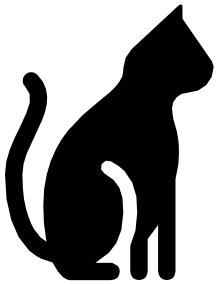


Objektorientierung
Klassen- und Objektdiagramme

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Klasse

Cat
- name : String - age : int - positionX : double - positionY : double
+ Cat(name : String, age : int) + move(x : double, y : double) + getName() : String + getAge() : int



Objekt

Paula : Cat
name = „Paula“ age = 3 positionX = 20.3 positionY = 10.42

Unified Modeling Language (UML)
Klassen- und Objektdiagramme

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Klasse

Employee
- pNr : Integer - name : String - dateOfBirth : String
+ Employee(nr : Integer, name: String) + getName(): String + getDateOfBirth(): String

一个和其他对象

Objekt

maxMeier : Employee
pNr = 2776 name = "Max Meier" dateOfBirth= "17.01.80"

Kurzform

Employee

Objektorientierung
Klassen- und Objektdiagramme

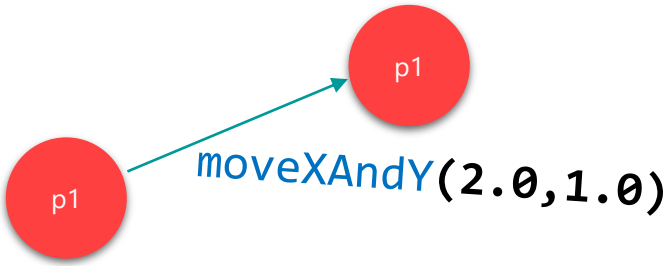
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Klasse

Point
- x : double - y : double
+ setXAndY(x : double, y : double) + moveXAndY(dx: double, dy : double)

Objekt

p1 : Point
x = 1 y = 3



Objektorientierung
Objekte: Instanzen einer Klasse

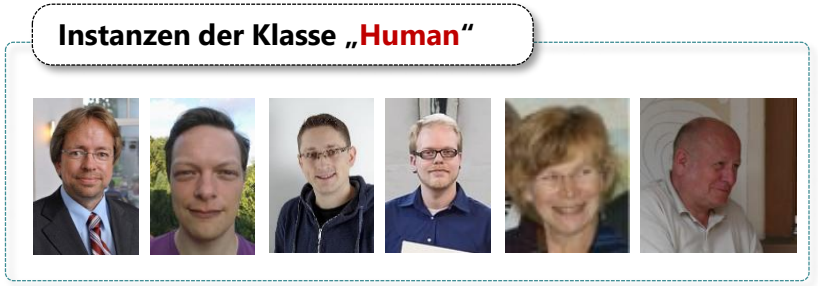
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

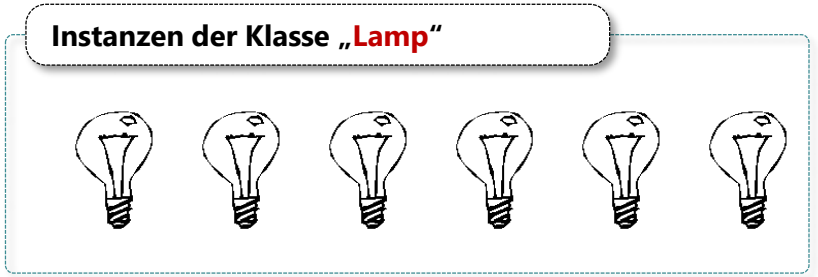
Objekte einer Klasse werden auch als Instanzen dieser bezeichnet

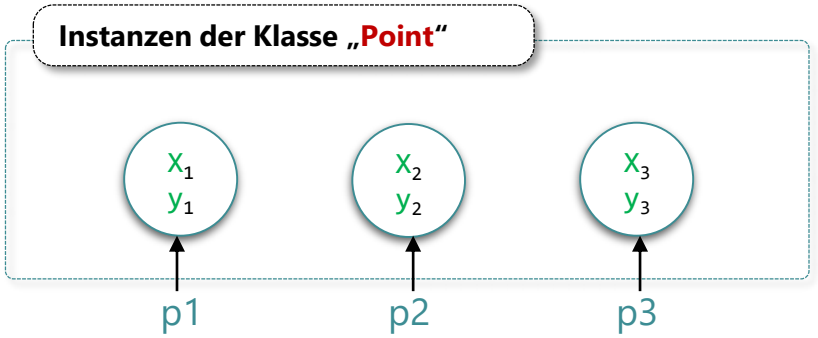
- Instanzen sind konkrete Exemplare von Klassen
- Instanzen existieren im Arbeitsspeicher (zur Laufzeit)
- Klassen sind nur **abstrakte Definitionen**

Beispiele:

- Human vs. „Ich“
- Lamp vs. „Diese Deckenlampe“
- Point vs. „p1“







Objektorientierung
Erzeugung von Objekten

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Instanzen von Klassen werden über **new** erzeugt (wie Arrays)

Bisher: `double[] p0 = new double[2];`

- erzeugt bzw. konstruiert und initialisiert ein neues Array
- liefert Adresse des neu erzeugten Arrays

Jetzt auch: `Point p1 = new Point();`

- erzeugt und initialisiert ein neues Objekt vom Typ Punkt
- liefert Adresse des neu erzeugten Punktes
- Reserviert einen Speicherbereich für die Attribute des Objekts

`p0` und `p1` werden *Referenzvariablen* (参考变量) *genannt*

`p0` und `p1` sind jeweils die Namen der Variablen. Über sie kann auf die Objekte zugegriffen werden.

Objektorientierung
Objekte und Arbeitsspeicher

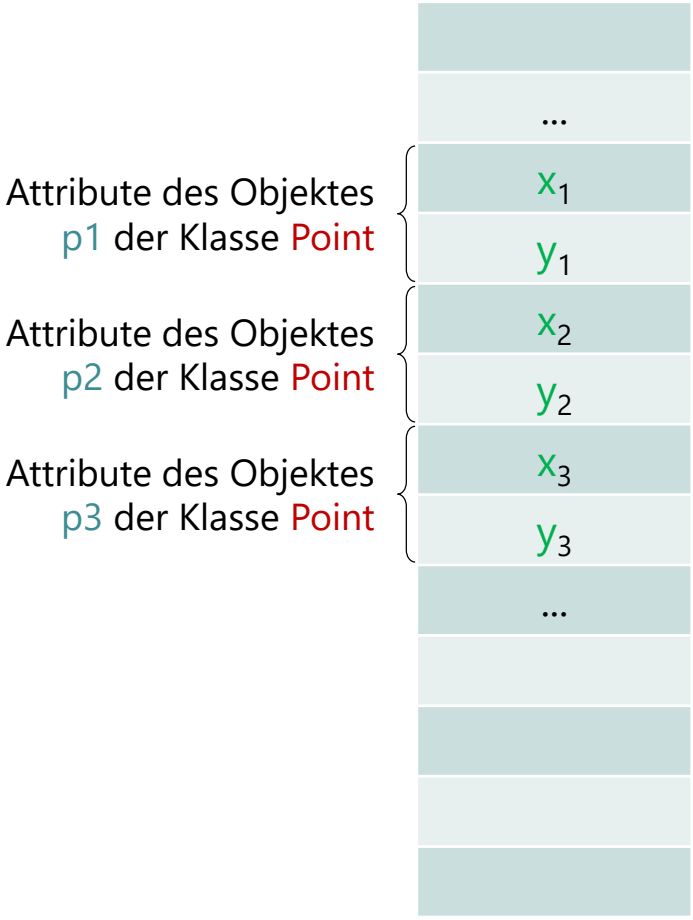
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Jedes Objekt besitzt einen **eigenen Speicherbereich für seine Attribute**

- Jedes Objekt kann also seine Attribute individuell verändern

Wie greift ein Objekt auf seine Attribute zu?

- Anders gefragt: Woher kennt es seine eigene Adresse?
- Eigene Adresse bezeichnet man auch als „Identität“
- Identität: siehe nächste Folie

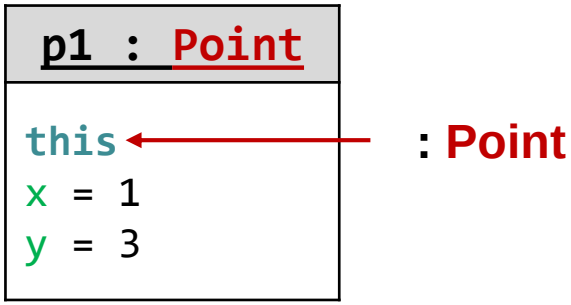


Objektorientierung
Identität von Objekten

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

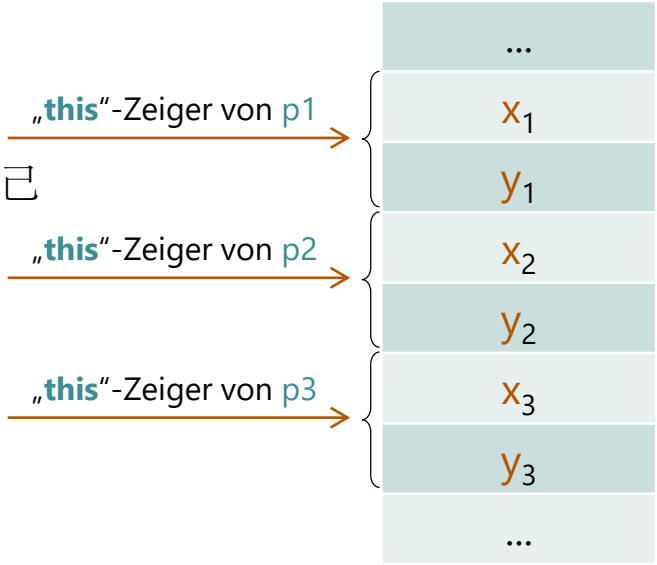
Jedes Objekt besitzt eine eigene **Identität**

In Form eines **Zeigers** auf den eigenen Speicherbereich



Der sog. „`this`“-Zeiger

- Analog zum „ich“ bei Menschen
- Quasi ein **Verweis auf sich selbst** 表示他自己
- Ein **implizit vorhandenes** Attribut
- Vom Typ der Klasse 已有的，现有的特征



Objektorientierung
Interaktion zwischen Objekten

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Objekte können über Nachrichten miteinander interagieren

Ablauf:

- Objekt x versendet Nachricht n an Objekt y
- Objekt x ruft dazu eine Methode m der Schnittstelle des Objekts y auf
- Die Spezifikation der Methode m legt fest,
  - ob Eingabe-Parameter übergeben werden und
  - ob ein Ausgabeparameter zurückgegeben wird

Darüber hinaus gilt:

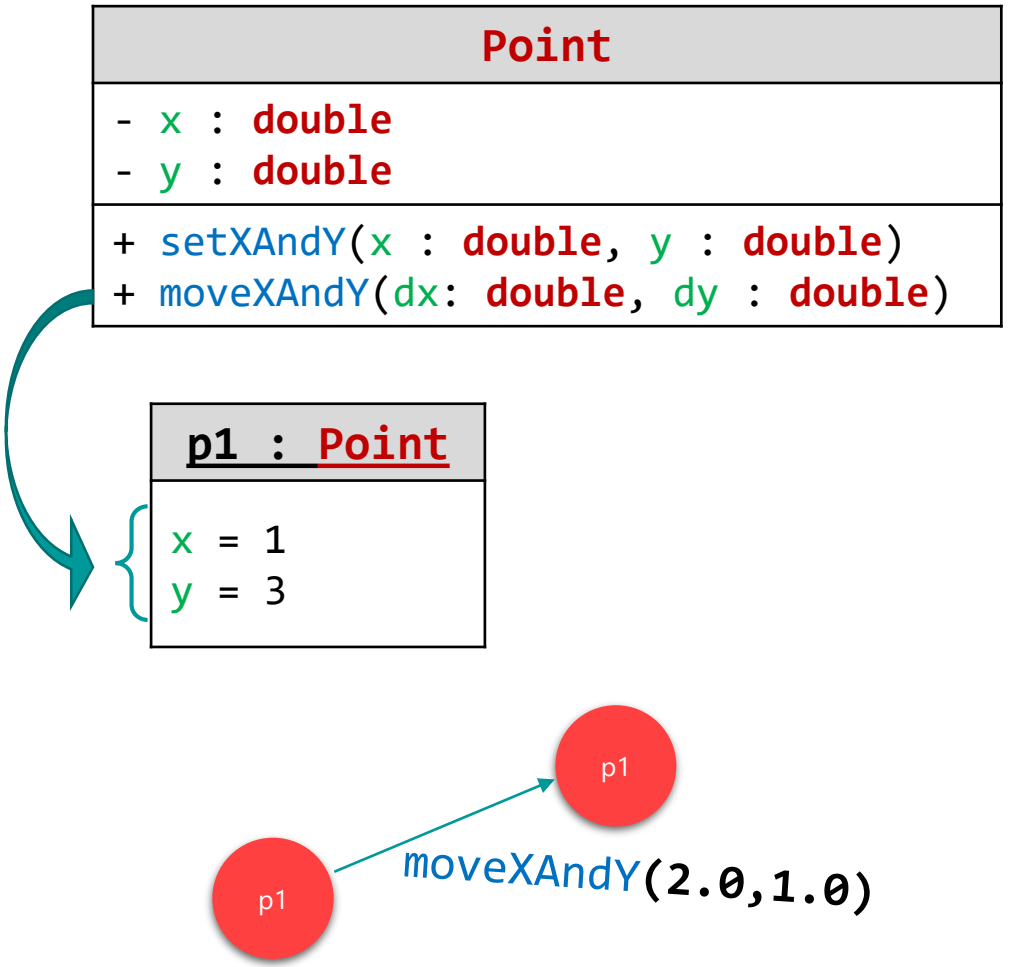
- Ein Objekt kann sich auch **selbst eine Nachricht schicken** (so können auch private Methoden aufgerufen werden)
- Nachrichten **entsprechen i. d. R. Prozedur- oder Funktionsaufrufen** im Kontext einer aufrufenden Methode

Objektorientierung
Methoden

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Methoden können den Zustand eines Objektes verändern:

- Also die Werte der Attribute verändern
- Beispielsweise verändert „moveXAndY“ die Werte der Attribute x und y
- Methoden *können* Eingabeparameter besitzen



Objektorientierung
Beziehung zwischen Objekten

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Objekte können in Beziehung / Relation zu einander stehen

- Firma f
- steht in Beziehung „Beschäftigungsverhältnis“ zu
- Person p

Die Beteiligten einer Beziehung nehmen dabei je eine Rolle ein

- Rolle der Firma f: Arbeitgeber
- Rolle der Person p: Arbeitnehmer

Ein Objekt kann mit mehreren Objekten in Beziehung stehen

- Firma f
- steht in Beziehung „Beschäftigungsverhältnis“ zu
- Person p_1 und
- Person p_2

Objektorientierte Programmierung

Objektorientierung in Java

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Schlüsselwort: public
siehe Folie „Geheimnisprinzip"

Jede (öffentliche) Klasse wird in einer separaten Datei abgelegt

- Dateiname entspricht dem Klassennamen
- Konvention - **Großschreibung** für Klassennamen
  - Beispiele: `Point`, `Mensch`, `Lampe`, `Bus`, `String`, …

对象名称大写

Erinnerung: *Klein beginnen und Camel Case weiter bei …*

- primitiven Datentypen `byte, short, boolean`
- Attributnamen `myInt, busNr`
- Methodennamen `moveXAndY()`

行为，特征，数据类型小写

Objektorientierung in Java
Klassen in Java

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Schlüsselwort: **`class`**

Syntax:

```
1  class <Klassenname> {
2      <Attribute und Methoden>
3  }
```

Beispiele:

- "Point.java":

```
1  class Point {
2      …
3  }
```

- "Human.java":

```
1  class Human {
2      …
3  }
```

Objektorientierung in Java
Attribute

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Syntax: `<Typ> <Name>;`

`<Typ>` Primitiver Datentyp, Array oder Name einer Klasse

Beispiel:

```
1  class Point {
2      double x;
3      double y;
4  }
```

Attribute der Klasse Punkt

Zugriff auf Attribute über `<Objektreferenz>.<Attributname>`

- Nur innerhalb der selben Klasse oder wenn die Zugriffssteuerung es zulässt
- Ansonsten werden sog. Getter und Setter Methoden benötigt

```
1  Point p1 = new Point();
2  p1.x = 1;  p1是行为的表示，x是新的特征
3  System.out.println("p1.x = " + p1.x);
```

Objektorientierung in Java
Verwendung von Attributen

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

```
1  class Point {
2     double x;
3     double y;
4
5     public static void main(String[] args) {
6
7         Point p1 = new Punkt();
8         p1.x = 1;
9         p1.y = 1;
10
11        Point p2 = new Punkt();
12        p2.x = 3;
13        p2.y = 4;
14
15        System.out.println("Punkt p1: (" + p1.x + "," + p1.y + ")");
16        System.out.println("Punkt p2: (" + p2.x + "," + p2.y + ")");
17    }
18
19 }
```

Achtung:
Die Methoden der Klasse Point sind in diesem Beispiel nicht enthalten. Außerdem sind die Klassenattribute öffentlich.

Achtung:
main-Methode ist Einstiegspunkt des *Programms*. Die ist *konzeptionell* nicht Teil der Klasse Punkt.

Objektorientierung in Java
Methoden

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

<Rückgabetyp> <Methodenname>(<Parameter(liste)>){…}

- Methoden geben einen Wert eines bestimmten Datentyps zurück
- Methoden können Eingabeparameter übergeben werden
- Methoden können Attribute lesen und verändern

Beispiele:

```
1  void setXAndY(double x, double y) {
2      ...
3  }
```

```
1  void moveXAndY(double dx, double dy) {
2      ...
3  }
```

Klassen und Methoden
Die Klasse Point


```
1  class Point {
2
3      private double x;
4      private double y;
5
6      void setXAndY(int x, int y) {
7          this.x = x;
8          this.y = y;
9      }
10
11     void moveXAndY(double dx, double dy) {
12         this.x += dx;
13         this.y += dy;
14     }
15 }
```

Klassen und Methoden
Aufruf von Methoden

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Wie Attribute werden Methoden eines Objektes über seine Referenzvariable angesprochen

Syntax: `<Objektreferenz>.<Methodenname>(...)`

Beispiele:

```
1 Punkt p1 = new Punkt();
2 p1.verschiebe(1.2, 3.4);
```

Achtung:
Gilt nicht für den Konstruktor!
Mit dessen Aufruf wird ein neues Objekt konstruiert.

Klassen und Methoden


```java
class Point {
	private double x;
	private double y;

	void setXAndY(double x, double y) {
		this.x = x;
		this.y = y;
	}
	void moveXAndY(double dx, double dy) {
		this.x += dx;
		this.y += dy;
	}
	public static void main(String[] args) {
		Point p1 = new Point();
		Point p2 = new Point();
		p1.x = 1;
		p1.y = 2;
		p2.x = 3;
		p2.y = 4;

		System.out.println("Punkt p1: (" + p1.x + "," + p1.y + ")");
		System.out.println("Punkt p2: (" + p2.x + "," + p2.y + ")");

		p1.setXAndY(3.3, 0.7);
		p1.moveXAndY(1.2, 3.6);

		System.out.println("Punkt p1: (" + p1.x + "," + p1.y + ")");
		System.out.println("Punkt p2: (" + p2.x + "," + p2.y + ")");
	}
}
```

Es werden 2 Instanzen der Klasse Point erzeugt (p1 und p2)

Den Attributen werden Werte *direkt* zugewiesen und dann ausgegeben

Auf dem Objekt p1 werden `setXAndY` und `moveXAndY` aufgerufen

Alle Werte werden erneut ausgegeben

Es haben sich nur die Attribute von p1 geändert

Klassen und Methoden
Überladen von Methoden

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Mehrere Methoden gleichen Namens sind in einer Klasse **erlaubt**

- Prinzip der sog. **Überladung**
- Gleicher **Rückgabetyp**, aber unterschiedliche Parameter

Beispiel: Einzelne Klasse mit folgenden Methoden

```
void print(int i){…}
void print(int j){…}
void print(String s){…}
void print(double d){…}
```

Fehler: gleicher Datentyp des Übergabeparameters

Aufrufe:

```
print(1);          // Aufruf von void print(int i);
print("Hallo");    // Aufruf von void print(String s);
print(2.098);      // Aufruf von void print(double d);
```

Objektorientierte Programmierung

Konstruktoren

Konstruktoren
Wichtiges Prinzip: Konstruktion

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Instanzen können bei der Erzeugung **automatisch initialisiert** werden

- Aufruf von speziellen Methoden durch Java
- Geschieht bei der Erzeugung mittels **new**

Diese speziellen Methoden heißen Konstruktoren

- Werden automatisch bei der Erzeugung (durch **new**) aufgerufen
- Bringen Instanzen in einen „sicheren“ initialen Zustand

In anderen Programmiersprachen gibt es auch Destruktoren

- z. B. zum Freigeben von Speicher etc.

Konstruktoren
Umsetzung in Java

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Konstruktoren: Methoden mit speziellem Namen

- Name der Methode: **Klassenname (Groß geschrieben)**
- **Rückgabetyp: Keiner**, nicht einmal `void`
- Konstruktor sollte alle Datenfelder initialisieren (guter Stil)

Syntax: `<Klassenname>(<Parameter(liste)>) {...}`

Beispiele:
```
Point() {...}
Point(double x, double y) {...}
```

Eine Klasse kann mehrere Konstruktoren haben

- Gleiches Prinzip wie beim Überladen von Methoden

Ein Konstruktor kann auch einen anderen Konstruktor aufrufen

- Syntax: `this(<Parameter(liste)>);`

Konstruktoren
Beispiel

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Erzeugung von zwei Point-Instanzen in main

- Kein Parameter an den Konstruktor
- Aufruf des parameter-losen Konstruktors aus Zeile 6
- Dieser ruft Konstruktor in Zeile 10 auf

- Zwei double-Parameter
- Aufruf des Konstruktors in Zeile 10

```
1  public class Point{
2
3      private double x;
4      private double y;
5
6      public Point(){
7          this(0.0, 0.0);
8      }
9
10     public Point(double x, double y){
11         this.x = x;
12         this.y = y;
13     }
14
15     public static void main(String[] args){
16         Point p1 = new Point();
17         Point p2 = new Point(1.0, 2.0);
18     }
19 }
```

Konstruktoren
Standard- & Default-Konstruktor

Parameterloser Konstruktor (auch „Standard-Konstruktor“)

- Eingangsparameter: keine
- Explizite Deklaration durch Programmierer*innen

Default-Konstruktor

- Implizit vorhandener parameterloser Konstruktor
- Wird nicht manuell von Programmierer*innen deklariert
- Nur vorhanden, wenn kein anderer Konstruktor der Klasse existiert

Konstruktoren
Standard- & Default-Konstruktor - Beispiel


```
1  public class Point {
2
3      private double x;
4      private double y;
5
6      public Punkt(double x, double y) {
7          this.x = x;
8          this.y = y;
9      }
10
11     public static void main(String[] args) {
12         Point p1 = new Point();
13         Point p2 = new Point(1.0, 2.0);
14     }
15 }
```

Fehler:
Es existiert kein parameterloser Konstruktor

Objektorientierte Programmierung

Geheimnisprinzip

Geheimnisprinzip
Zentrale Eigenschaft von Objekten

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Geheimnisprinzip: Objekte kapseln ihre „Interna“, d. h.

- ihren Zustand (Belegung der Attribute)
- die Implementierung
  - ihrer Zustandsrepräsentation (Details zu ihren Attributen)
  - ihres Verhaltens (der Implementierung ihrer Methoden)

Objekte sind einzig über ihre Schnittstelle zugänglich

- d. h. über die der „Außenwelt“ zur Verfügung gestellten Methoden
- Attribute (und „interne“ Methoden) sind privat
- Wichtige Schlüsselworte
  - `public`: Zugriff „von außen“ möglich (öffentlich)
  - `private`: Zugriff „von außen“ **nicht** möglich (privat)

Geheimnisprinzip
Private Attribute

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

```java
public class Point {

    private double x;
    private double y;

    public void setXAndY(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return this.x;
    }

    public double getY() {
        return this.y;
    }

    public void moveXAndY(double dx, double dy) { … }
}
```

„main-Methode" befindet sich **außerhalb der Klasse Punkt**

```java
public class MeinProgramm
    public static void main(String[] args) {
        Punkt p = new Punkt();
        p.setXAndY(1, 2);
        System.out.println("Punkt p: (" + p.x + "," + p.y + ")");
        System.out.println("Punkt p: (" + p.getX() + "," + p.getY() + ")");
    }
}
```

Objektorientierte Programmierung

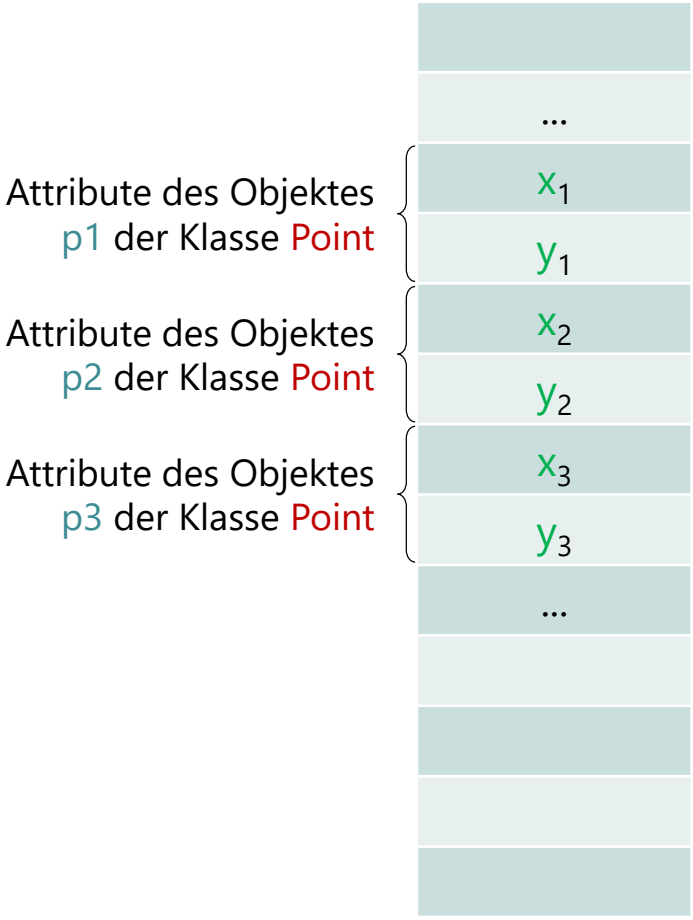
Klassenattribute und -methoden VS. Objektattribute und -methoden

Objektattribute und -methoden
Einmal pro Instanz

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Objektattribute und -methoden

- Bestimmen Zustand und Verhalten **eines individuellen** Objekts
- **Stehen erst nach Instanziierung des Objektes zur Verfügung**
- Sie **existieren einmal** pro Instanz
- Variablen und Konstanten können in unterschiedlichen Instanzen unterschiedlich belegt sein
- Objektmethoden haben zugriff auf
  - Objektattribute und –methoden des jeweils eigenen Objektes und auf
  - Klassenattribute und -methoden



Klassenattribute und -methoden
Motivation

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Manchmal praktisch, dieses Modell zu durchbrechen

- Main-Class:
  - Enthält Programmeinstiegspunkt (Main Methode)
  - Instanzen der Klasse nicht gewünscht/notwendig

Beispiel: Mathematische Funktionen

- Wurzel, Sinus, Kosinus, etc.
- Primitive Datentypen sind keine Objekte
  - ~~`double d = 1.0; d.sinus();`~~ → Funktioniert nicht
  - `new Sinus(1.0).getValue();` → Zu großer Overhead
- Lösung für dieses Beispiel: `Math.sin(1.0);`
  - Aber wie geht das, ohne ein Objekt von Math zu instanziieren?

Klassenattribute und -methoden
Das Schlüsselwort static

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Schlüsselwort: `static`

- Als **`static`** deklarierte Variablen und Methoden werden Klassenvariablen bzw. Klassenmethoden genannt
- Diese stehen bereits zu Beginn der Ausführung des Programmes zur Verfügung
  - Können verwendet werden, ohne dass zuvor explizit ein Objekt einer Klasse erzeugt werden muss
  - Sind zudem unabhängig von möglichen Instanzen der Klasse
- **Statische Elemente können nur andere statische Elemente verwenden**
  - Aufruf anderer *statischer* Methoden oder Verwenden *statischer* Attribute
  - können keine „`this`"-Referenz nutzen (keine Instanz verfügbar)
  - Es gibt keinen Zugriff auf Objektattribute und -methoden

Klassenattribute und -methoden
Statische Methoden und Attribute

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Statische Methoden (Klassenmethoden)

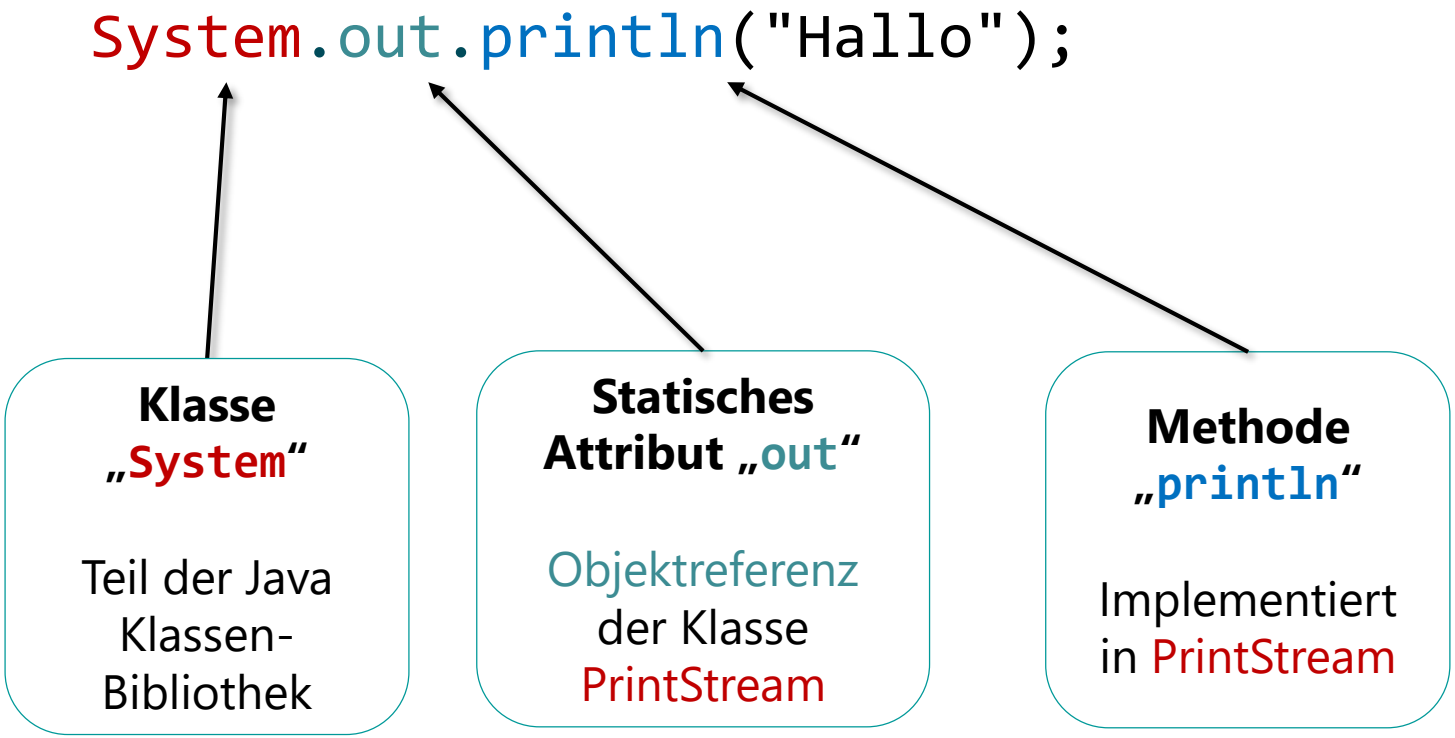
- Definition:
```
static void staticMethod(...) {...}
```
- Verwendung:
```
<Klassenname>.<Methodenname>(<Parameterliste>);
```

Statische Attribute (Klassenattribute)

- Definition:
```
static <Typ> <statischesAttribut>;
```
- Verwendung:
```
<Klassenname>.<statischesAttribut>;
```

Klassenattribute und -methoden
Beispiel: Standardausgabe

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK



Klassenattribute und -methoden
Beispiel statisch/nicht-statisch

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

```java
public class StaticTest {

    private static int statisch = 1;
    private int nichtStatisch = 2;

    public void nichtStatischeMethode() {
        statisch++;                     // OK
        nichtStatisch++;                // Ok
        statischeMethode();             // Ok

        this.nichtStatisch++;           // Ok
        this.statisch++;                // Ok, aber unschön
        this.statischeMethode();        // Ok, aber unschön
    }

   public static void statischeMethode() {
        statisch++;                     // OK
        nichtStatisch++;                // Nicht ok
        nichtStatischeMethode();        // Nicht ok

        this.nichtStatisch++;           // Nicht ok
        this.statisch++;                // Nicht ok
        this.statischeMethode();        // Nicht ok
    }
}
```

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Kontakt

Raphael Allner, M. Sc.
Wissenschaftlicher Mitarbeiter
Institut für Telematik

Universität zu Lübeck
Ratzeburger Allee 160
23562 Lübeck

https://www.itm.uni-luebeck.de/mitarbeitende/raphael-allner.html