Javakurs 2011 – Objektorientierung

Objektorientierte Programmierung I

Martin Kresse
&
Katrin Lang

21. März 2011

Technische Universität Berlin

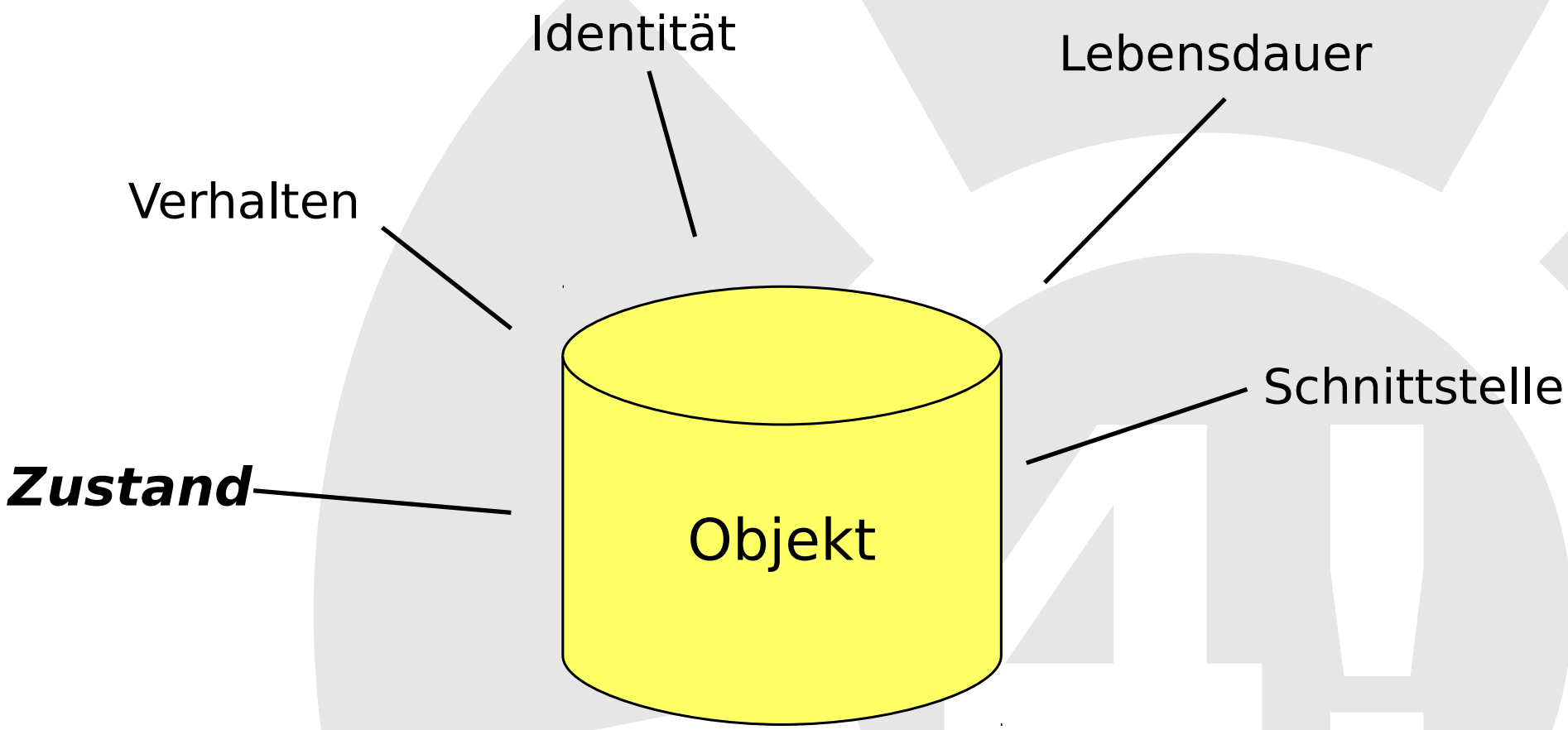
Was wir bereits kennen

return int Parameter long Variablen
Array boolean short switch else
for byte do main continue while
Rückgabewert double break import
float Datentyp public default if
void char Methode case Operator

Warum Objektorientierung?

- separation of concerns (Trennung der Aufgabenbereiche)
- flexible Kombinierbarkeit
- höhere Abstraktionsebene --> Verringerung der Komplexität
- Wiederverwendbarkeit
- ...

Worum geht es heute?



Attribute speichern Zustand

Attribute speichern Zustand

Klasse vs. Instanz

Klasse Mensch = Bauplan

Instanzen = Objekte

Klassen definieren Attribute

```
1 class Human {
2     String name;
3     int age;
4 }
```

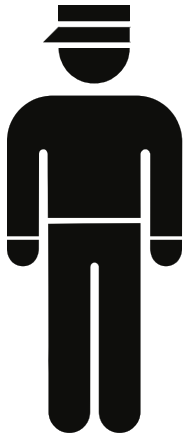
Klassenname

Attribute / Instanzvariablen

- Klasse = Bauplan für Objekte
- beschreibt Eigenschaften aller Objekte
- Objekt = Instanz einer Klasse
- Konvention zur Schreibweise:
  - Klassennamen groß
  - primitive Datentypen klein
  - Variablennamen klein

Erstellen von Objekten

```
1 class Human {
2     String name;
3     int age;
4 }
```

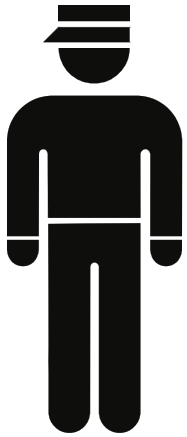


```
1 Human martin = new Human();
2
3
4
5
6
7
```

zeigt auf eine Instanz der Klasse *Human*

Zugriff auf Attribute

```
1 class Human {
2    String name;
3    int age;
4 }
```



```
1 Human martin = new Human();
2
3 martin.name = "Martin Kresse";
4 martin.age = 31;
5
6 System.out.println(martin.name);
7 System.out.println(martin.age);
```

Arbeiten mit mehreren Klassen

Listing 1: Human.java

```
1 class Human {
2     String name;
3     int age;
4 }
```

Listing 2: Lecturer.java

```
1  class Lecturer {
2      public static void main(String[] args) {
3          Human martin = new Human();
4
5          martin.name = "Martin Kresse";
6          martin.age = 31;
7
8          System.out.println(martin.name);
9          System.out.println(martin.age);
10     }
11 }
```

- genau eine Klasse pro Datei
- Dateiname muss identisch mit Klassennamen sein

Arbeiten mit mehreren Klassen

Listing 1: Human.java

```
1 class Human {
2     String name;
3     int age;
4 }
```

Listing 2: Lecturer.java

```
1  class Lecturer {
2      public static void main(String[] args) {
3          Human martin = new Human();
4
5          martin.name = "Martin Kresse";
6          martin.age = 31;
7
8          System.out.println(martin.name);
9          System.out.println(martin.age);
10     }
11 }
```

Abhängigkeiten werden erkannt, Klasse mit *main()* reicht aus

```
$ javac Lecturer.java
$ ls
Human.class     Human.java      Lecturer.class  Lecturer.java
$ java Lecturer
```

Methoden implementieren Verhalten

Methodendefinition

Listing 3: Human.java

```
1  class Human {
2     String name;
3     int age;
4
5     void birthday() {
6        age++;
7
8        if (age >= 18) {
9           // TODO: sex, drugs & rock'n'roll
10       }
11    }
12 }
```

Methodenaufruf

```
1 Human martin = new Human();
2
3 martin.name = "Martin Kresse";
4 martin.age = 31;
5 martin.birthday();
6
7 System.out.println("Alter: " + martin.age);
```

Ausgabe:

Alter: 32

Bedeutung von *static*

```
1 Human.birthday();
2 Human.age = 18;
3
4 Human dorianGray = new Human();
  // Attribut out ist static
6 dorianGray.birthday();
7 dorianGray.age = Math.min(24, dorianGray.age);
8 System.out.println("Alter:" + dorianGray.age);
```

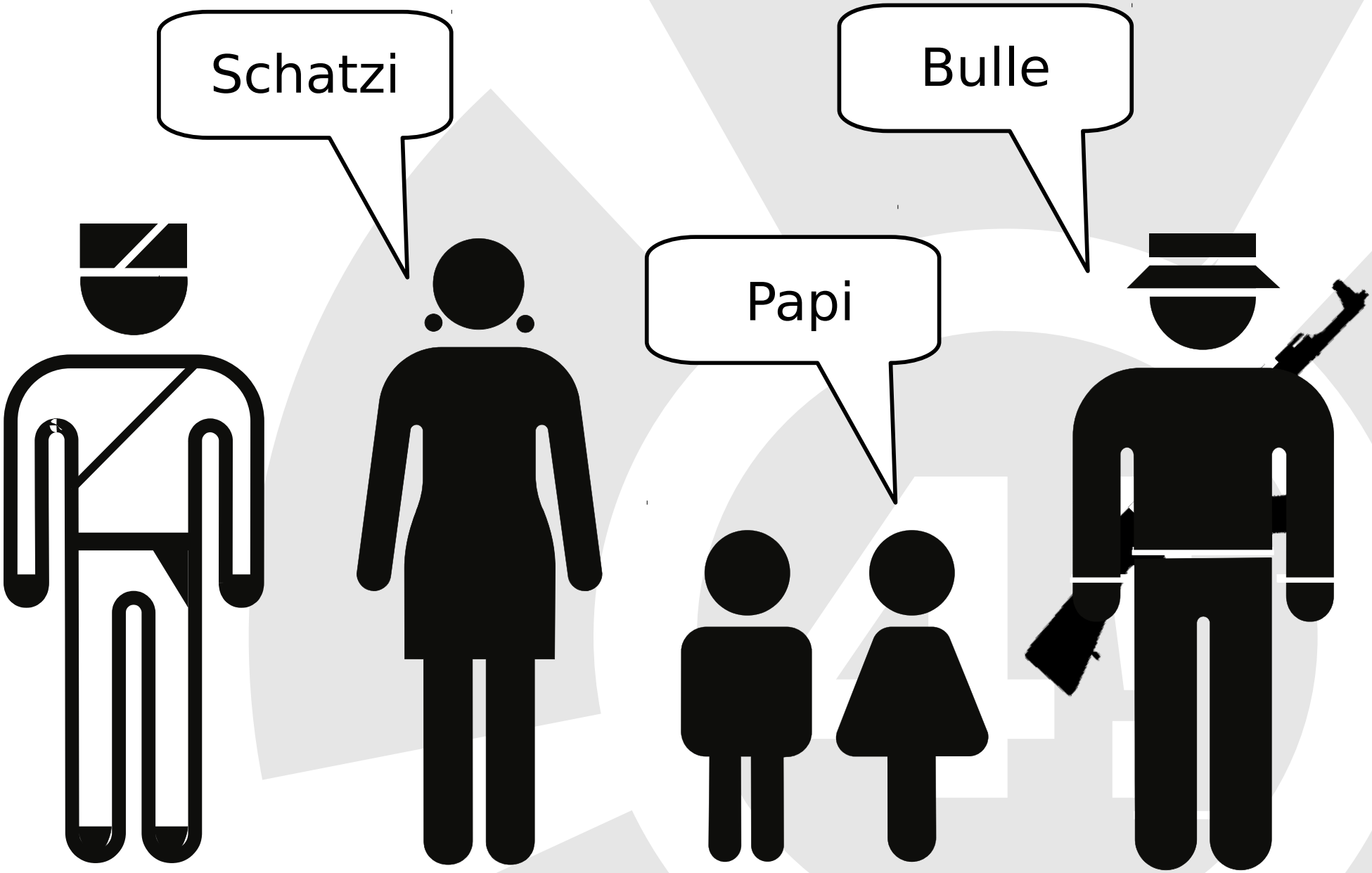
Methode *min()* ist *static*

statische Methoden	**nicht-statische Methoden**
• Klassen-Methoden	• Instanz-Methoden
• nur Zugriff auf Klassenvariablen	• Zugriff auf Klassen- und Instanzvariablen
• Können direkt auf Klasse aufgerufen werden	• erfordern Erstellung eines Objektes

Objekte haben eine Identität

Picture by pizzo, flickr.com

Variablen speichern Referenzen



Referenzen sind... anders

```
1 int a = 0;
2 int b = a;
3
4 a++;
5
6 System.out.println(a);
7 System.out.println(b);
```

```
1 Human a = new Human();
2 Human b = a;
3
4 a.age++;
5
6 System.out.println(a.age);
7 System.out.println(b.age);
```

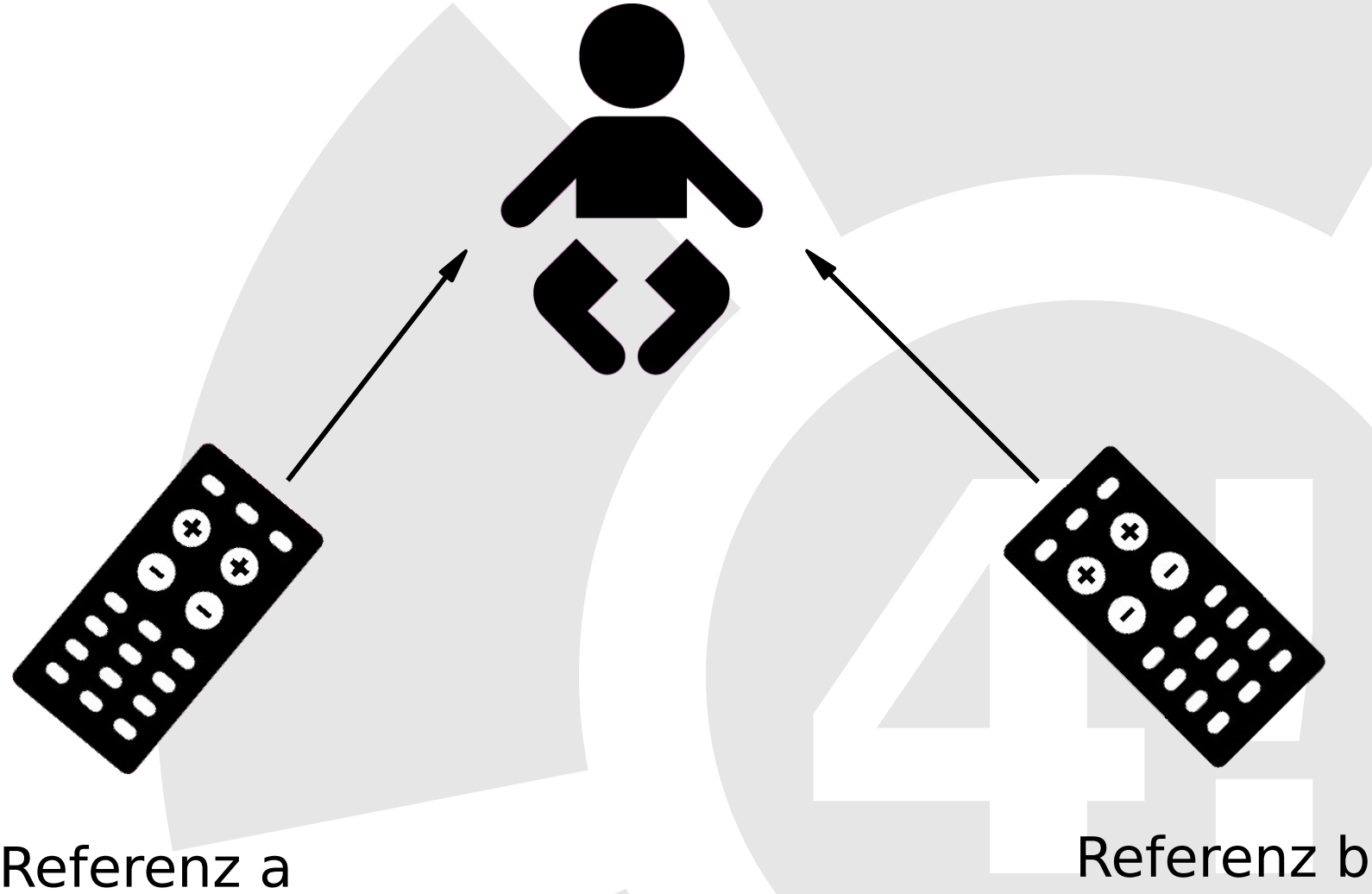
Ergebnis:

a: 1
b: 0

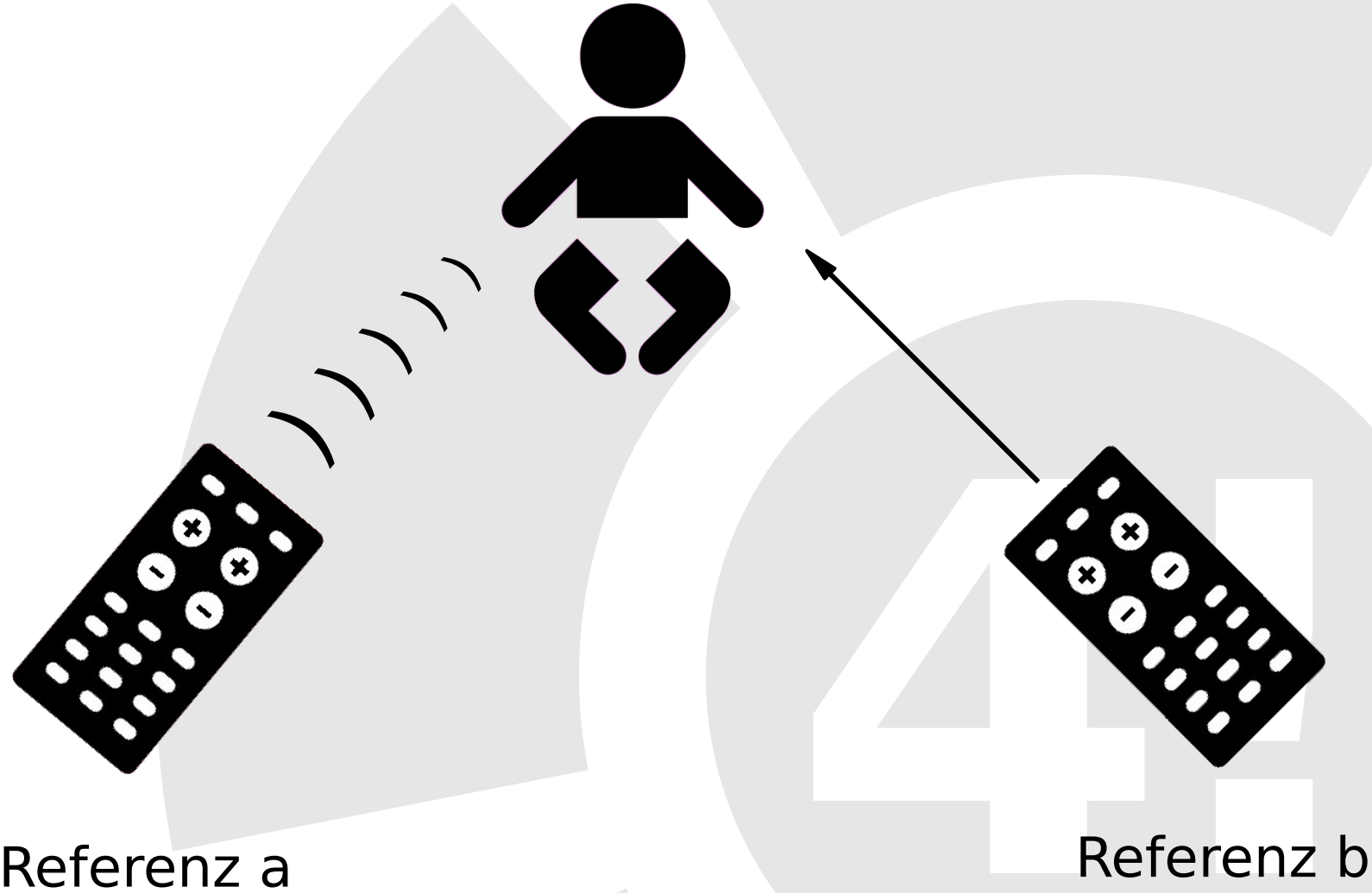
Ergebnis:

a.age: 1
b.age: 1

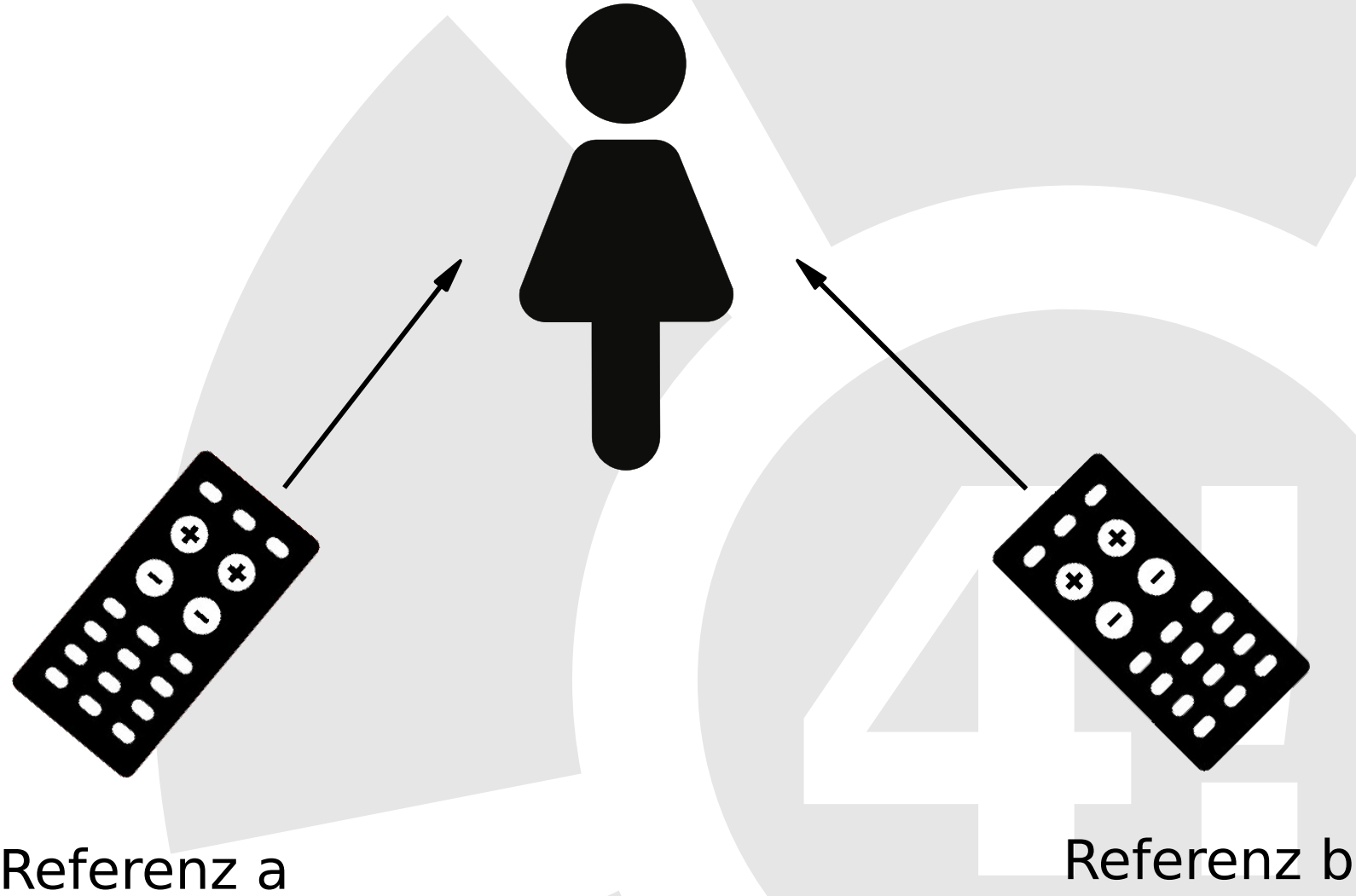
Referenzen



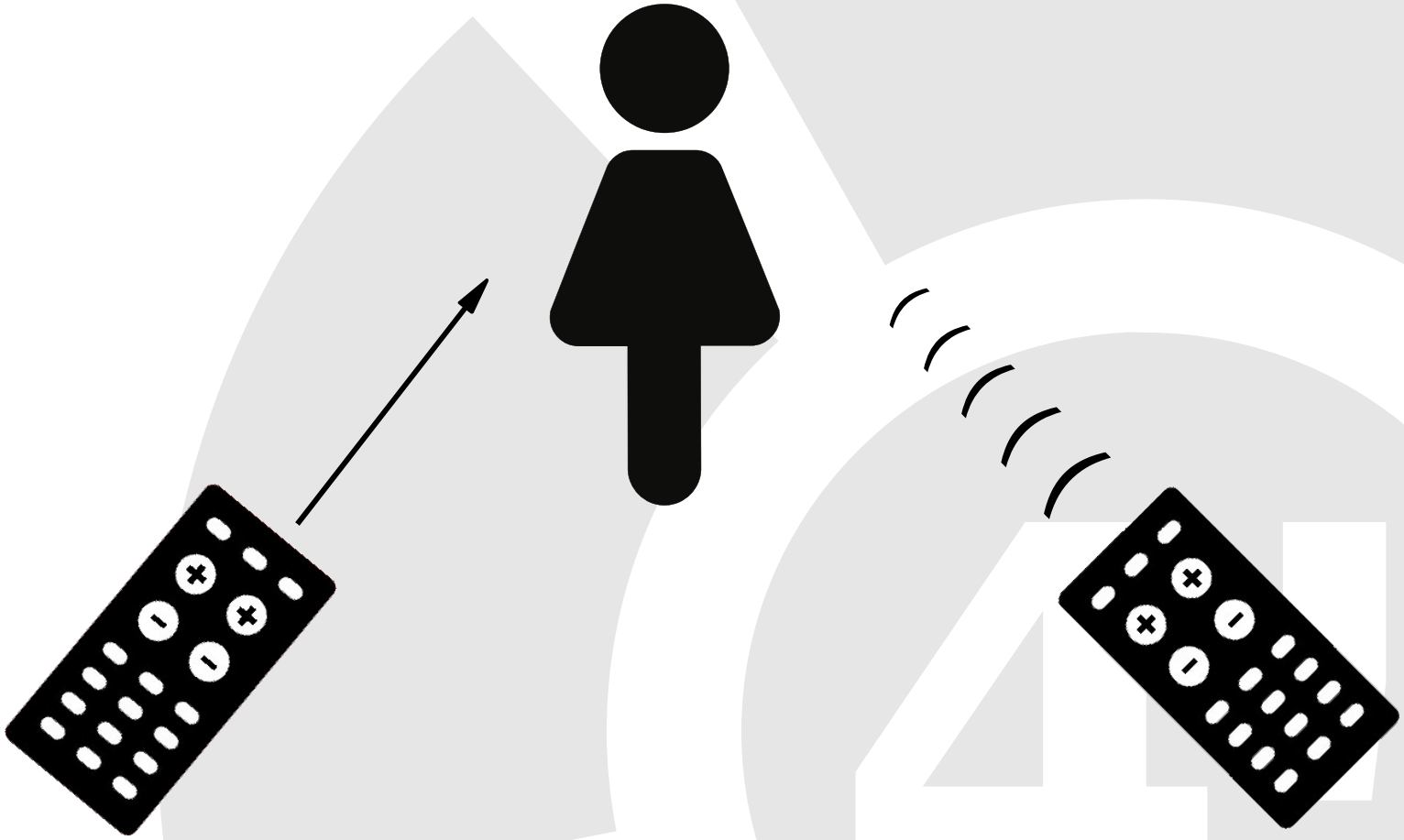
Referenzen



Referenzen



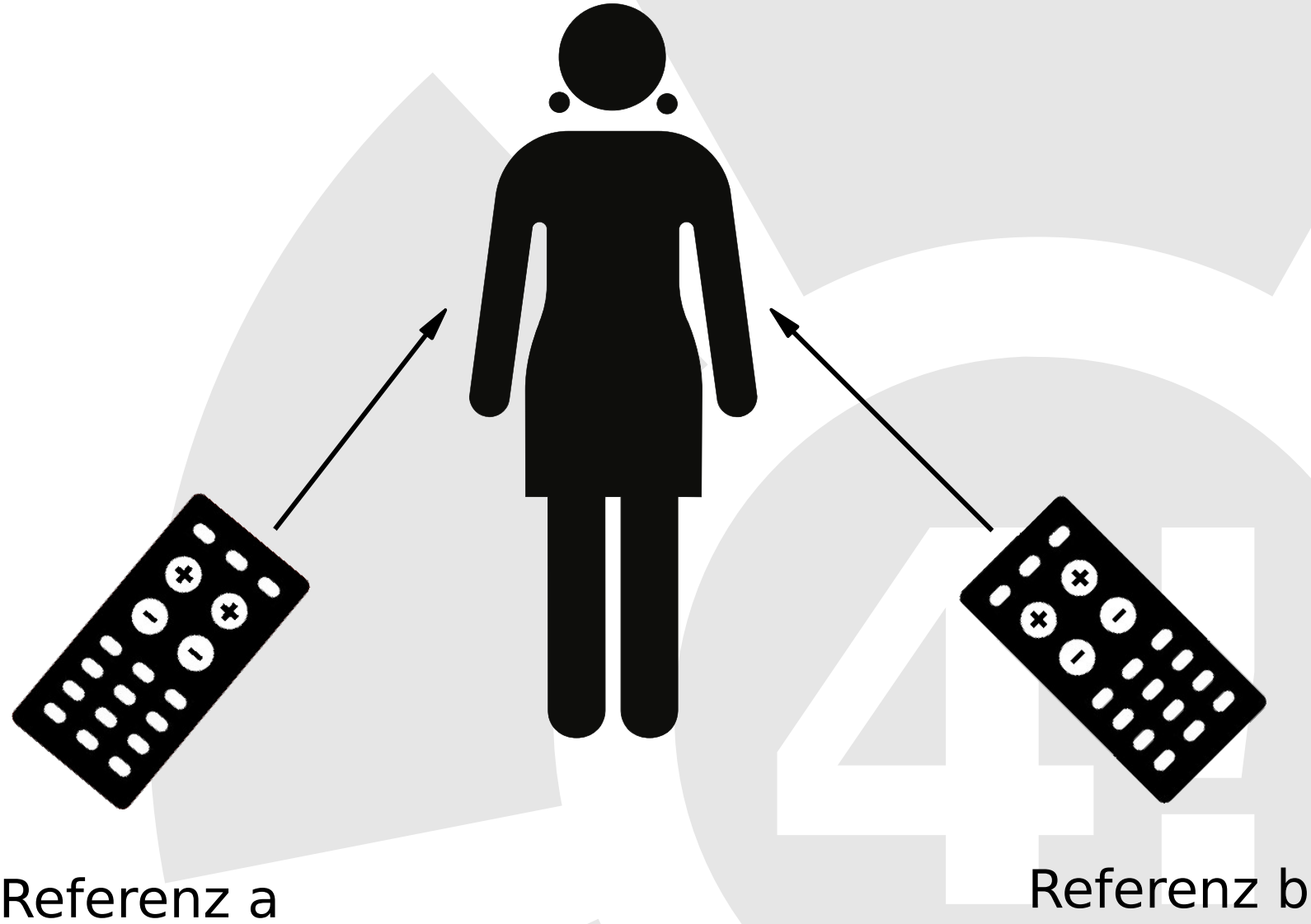
Referenzen



Referenz a

Referenz b

Referenzen



Konstruktoren

Standard-Konstruktor

Erzeugung eines Objektes:

```
Human martin = new Human();
```

Listing 4: Human.java *(mit Standardkonstruktor)*

```
1 class Human {
2    String name;
3    int age;
4
5    Human() {
6       name = null;
7       age = 0;
8    }
9 }
```

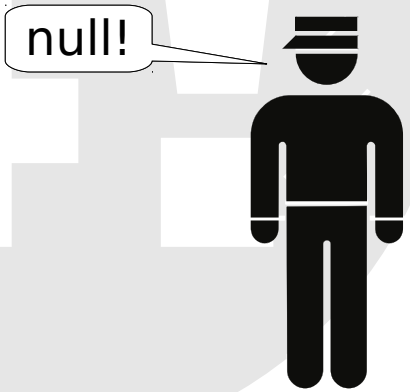

Vorsich

null-Referenzen

Listing 5: Human.java

```
1  class Human {
     [...]
23   void greet() {
24     if (name.contains("Dr."))
25       System.out.println("Guten Tag!");
26     else
27       System.out.println("Hi!");
28   }
29 }
```

```
Exception in thread "main"
 java.lang.NullPointerException
  at Human.greet(Human.java:24)
  at Human.main(Human.java:15)
```



Konstruktoren mit Parametern

Listing 6: Human.java *(mit parameterisiertem Konstruktor)*

```
1  class Human {
2     String name;
3     int age;
4
5     Human(String name, int age) {
6        this.name = name;
7        this.age = age;
8     }
      [...]
29 }
```

Erzeugung eines Objektes:

```
~~Human nobody = new Human();~~
Human karl = new Human("Karl-Theodor", 39);
karl.greet();
```


Vorsich

Konstruktoren mit Parametern

Listing 7: Human.java *(mit defektem Konstruktor)*

```
1  class Human {
2     String name;
3     int age;
4
5     Human(String name, int age) {
6        name = name;
7        age = age;
8     }
      [...]
29 }
```

Parameter überschatten gleichnamige Instanzvariablen

```
Human vitali = new Human("Dr. Eisenfaust", 39);
vitali.greet();
```

Ergebnis:

`java.lang.NullPointerException`

Überblick Konstruktoren

- Konstruktoren initialisieren Objekten
- Name = Klassenname
- Kein Rückgabewert
- Wird kein Konstruktor definiert, existiert ein impliziter Default-Konstruktor ohne Parameter

Compiletime- und Runtime-Fehler

```
1  Human nobody;
2  nobody.birthday();
```

Compiletime Fehler

```
3  nobody = null;
4  nobody.birthday();
```

Runtime Fehler

```
5  Human noOne = nobody;
6  noOne.birthday();
```

Runtime Fehler

```
7  Human somebody = new Human();
8  somebody.birthday();
9  System.out.print(somebody.name.length());
```

```
10 Human martin = new Human("Martin", 31);
11 martin.birthday();
```

```
12 Human javaTutorOfTheYear = martin;
13 javaTutorOfTheYear.birthday();
```

Kapselung (Information Hiding)

Direkter Variablenzugriff

```
1 Human martin = new Human();
2
3 martin.age = 31;
4 martin.birthday();
5
6 System.out.println("Alter: " + martin.age);
```

Ausgabe:

Alter: 32

Illegaler Zustand

```
1 Human martin = new Human();
2
3 martin.age = -2;
4 martin.birthday();
5
6 System.out.println("Alter: " + martin.age);
```

Ausgabe:

Alter: -1

Getter- und Setter-Methoden

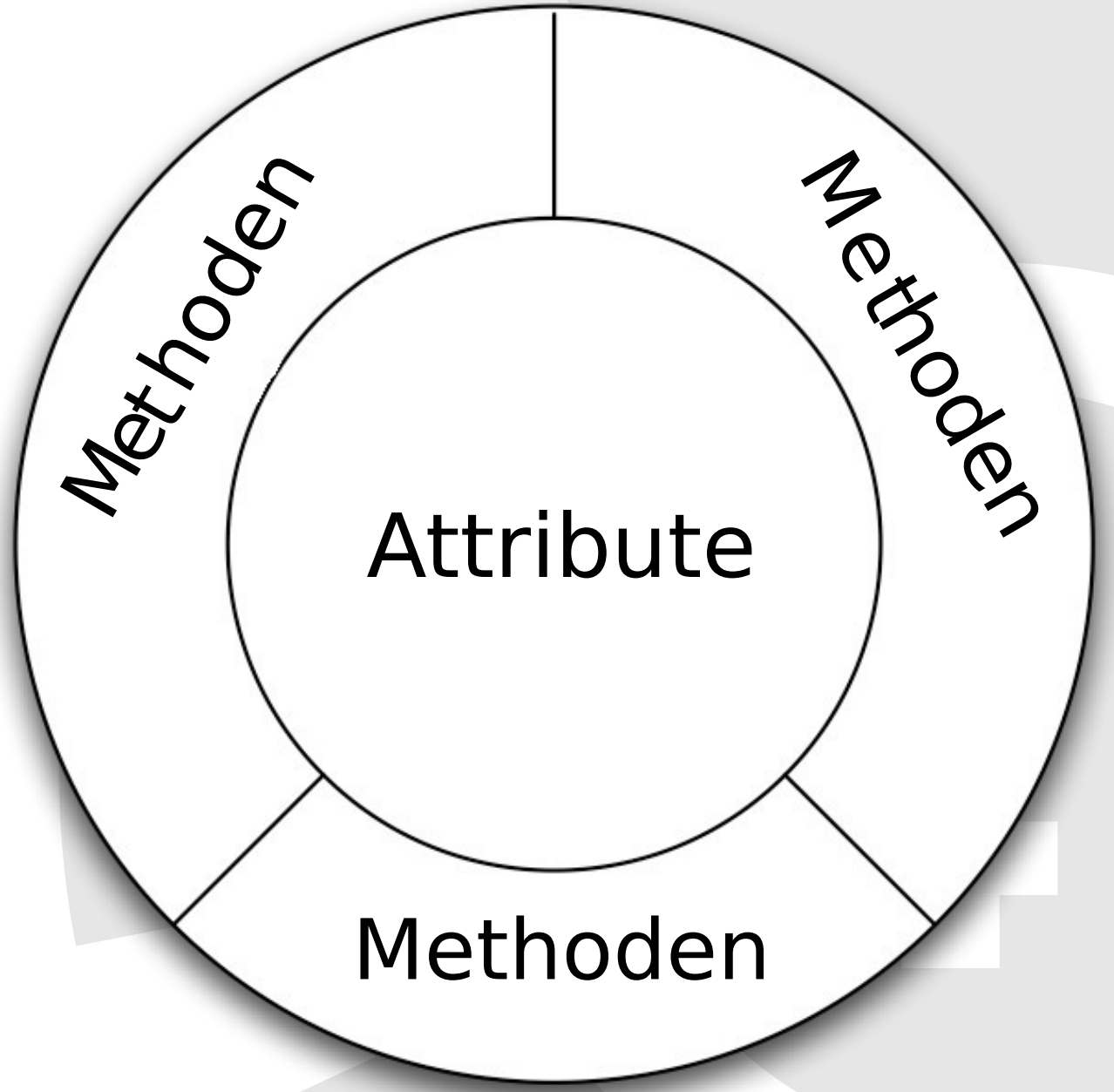
Listing 8: Human.java *(mit Getter- und Setter-Methoden)*

```
1  class Human {
2     private String name;
3     private int age;
4
5     public void setAge(int age) {
6        this.age = age;
7     }
8
9     public int getAge() {
10       return age;
11    }
12
```

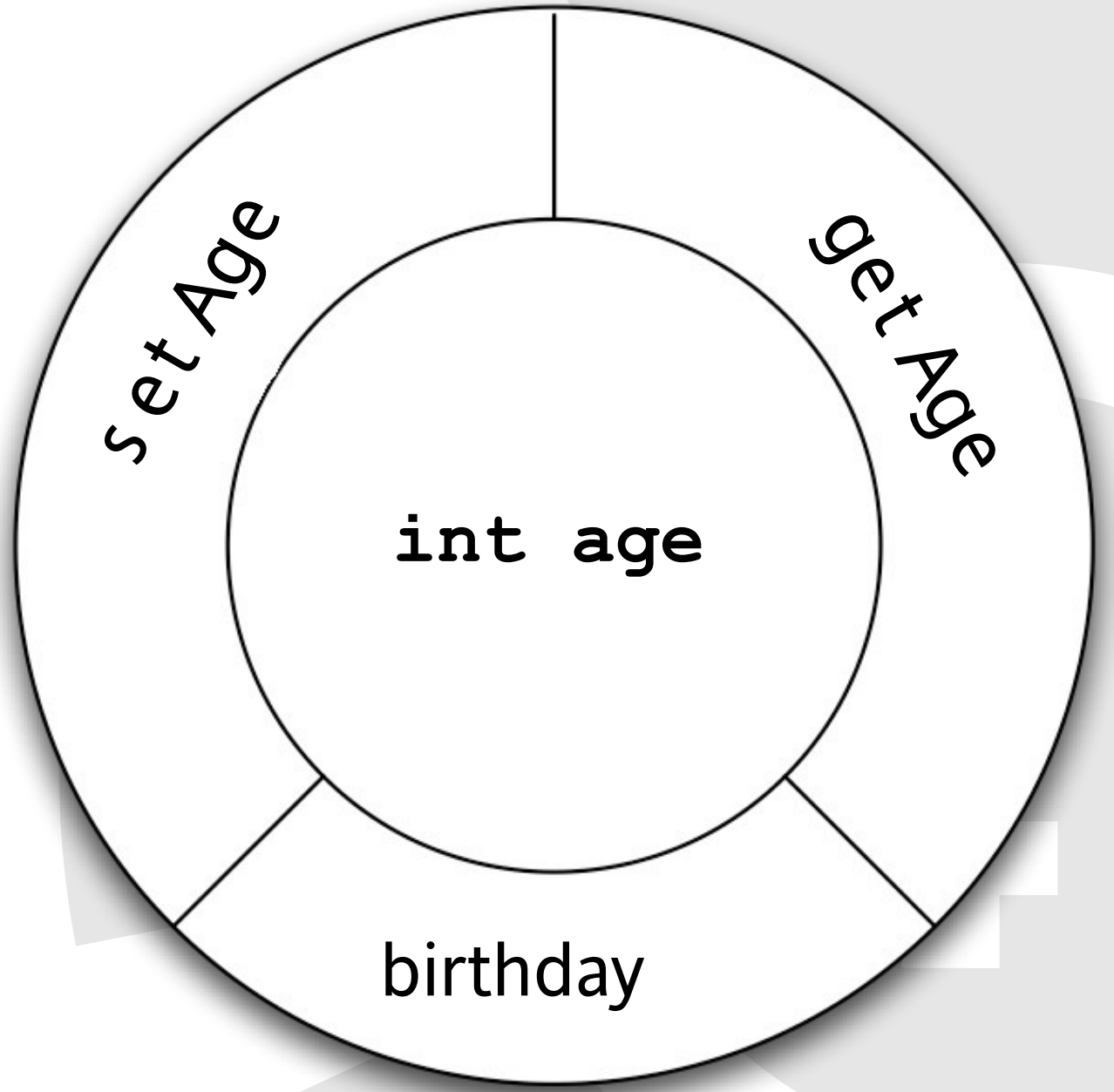
Instanzvariablen sind versteckt

Zugriffsmethoden sind öffentlich

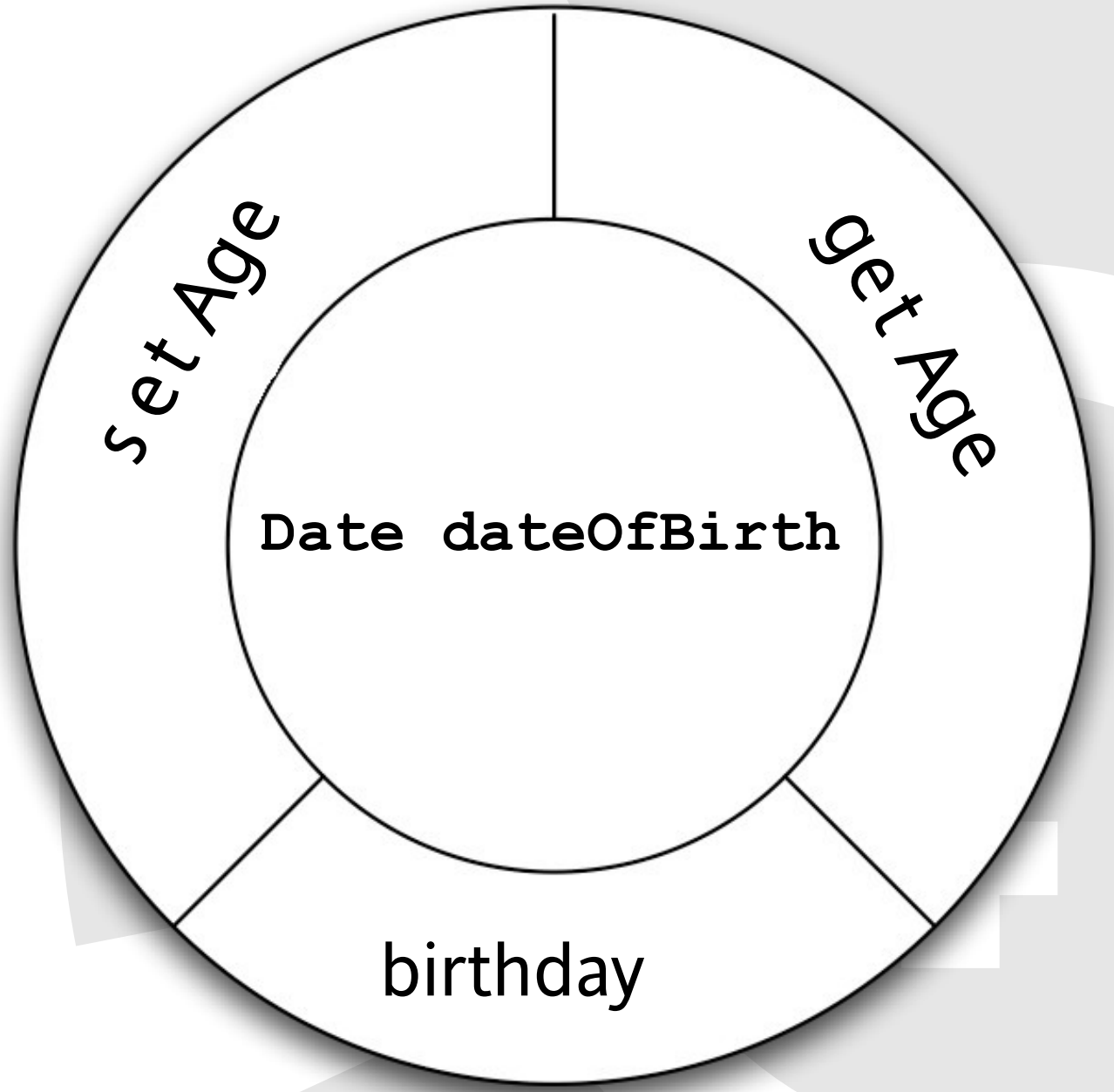
Die Idee von Kapselung



Die Idee von Kapselung



Die Idee von Kapselung



Getter- und Setter-Methoden

Listing 9: Human.java *(mit Geburtsdatum statt Alter)*

```
1  import java.util.Date;
2
3  class Human {
4     private String name;
5     private Date dateOfBirth;
6
7     public int getAge() {
8        Date today = new Date();
9        return today.getYear() - dateOfBirth.getYear();
10    }
11 }
```

Hausaufgabe:
getAge() korrigieren

Fragen?

Quellen

Piktogramme:

http://www.designofsignage.com

Fotos:

http://www.flickr.com
http://www.fotocommunity.com