Objektorientierung in Java

Java-Kurs 2007 - LE 5

Katrin Lang
langk@cs.tu-berlin.de

Martin Häcker
mhaecker@cs.tu-berlin.de

Motivation: Königsdisziplin (momentan?)
* So wird der meiste Code Programmiert
* Ganz ‚praktisch' wichtig für eure Zukunft!

Übersicht
Objektorientierung
Syntax
Etwas Design

Frage: Wer kennt schach, Wer spielt Schach, Wer spielt Schach gut? Wer spielt Schach wirklich gut?
Figuren Bewegen, Regeln, spielen, gut spielen, meisterhaft spielen?

Nichts neues hier, wissen alles schon. Nur etwas Syntax.
Nichts neues im Sinne von Schach.
Genau so ist‘s mit Objekten. Schnell erklärt, ein Leben lang dran gearbeitet.
OO: andere Strukturierung des Codes. Jeder machts sowieso, jetzt lernen anders darüber zu Denken.
Wie beim Schach, Intuition, Erfahrung, Phantasie, Kunst.

Easy to learn, hard to master.

Übergang: Motivation geben (wird viel zu oft vernachlässigt)
Erst Intuitiv, dann Code

Was ist ein Objekt

Der Begriff selber ist einfach, überall um uns denken wir in Objekten.
-> Objektorientierung ist Ideal um die reale Welt am Computer zu Simulieren
-> Das ist Reichweite und Grenze der Objektorientierung. (Keine Patentlösung)
-> Objektorientierung: Dieses Prinzip im Computer, Abbild der realen Welt

Überleitung: Wir betrachten ein gewöhnliches Objekt...

(pause) Hinweis bekommen: Aus Korrektheit ... für Informatiker ... auch Männer sind nur Objekte.

Gegensatz: Mensch (Klasse) <-> Person (Instanz)

Ich bin ein Mensch. Ich habe Haare, Augen, etc.

Der Mensch
Mensch ist unendlich kompliziert
* Reflexe / Atmen
* Gewollte koordination -> Schreiben
* Zusammenspiel der Muskeln
* Zusammenspiel der Zellen / kommunikation über hormone...

Instinktiv sehen wir Teile:
* Kopf, Bauch, Arme
* Top-Down!
* Beispiele für Denkweise

Überleitung: Erst wenn wir Teile genauer Betrachten (und erst dann) sehen wir weitere details

1. Kopf:
* Nase, Haare, Augebrauen, Augen
* Augen die Wieder aus teilen bestehen
* Vielleicht eine Sehschwäche haben

Beine erfüllen eine Funktion.

Uninteressant wie die erfüllt wird, nur das sie tun was verlangt ist.

Überleitung: Kann durchaus sein, das die Implementierung anders ist als von uns Gedacht...

... Prothesen enthalten

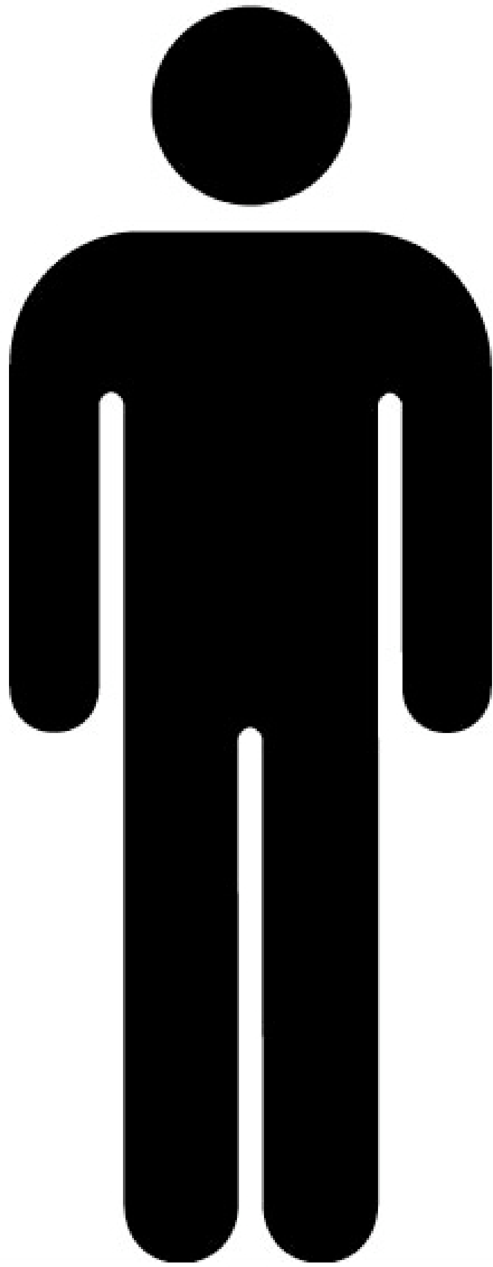
Denkt man Top-Down (Aus was besteht etwas)
Dann nimmt man _ERST_ wenn man die Details genauer untersucht
Deren „implementation“, deren „innereien“ wahr.

Im Ganzen ist nur wichtig „WAS“ etwas tut
Erst wenn man es für sich anschaut, wird interessant „WIE“ es das tut.

Das ist, was abstraktion ausmacht
-> Das ermöglich uns über Menschen / beliebige Dinge nachzudenken
  ohne das wir an tausenden details hängenbleiben und unser Hirn explodiert.

Jetzt wisst ihr, das ihr schon wisst was OO ist.

Übergabe: Dieses Beispiel jetzt in Code um die Syntax zu erklären.
Auf höherer Abstraktionsebene, aber konkreter, da in Code.

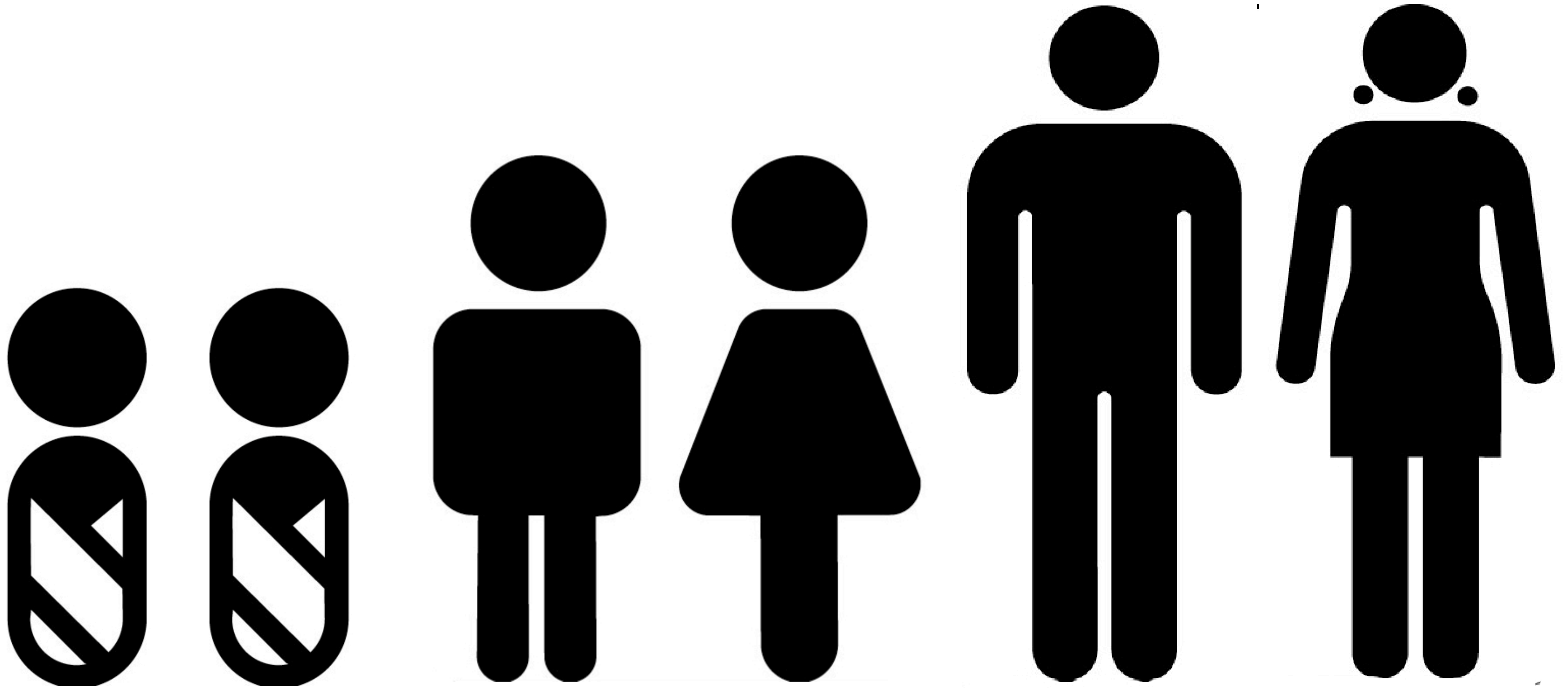


- Mensch verfügt über Fähigkeit zur Abstraktion
- Abbildung dieser Fähigkeit im Computer mit Hilfe der Objektorientierung
- Dazu werden wir systematisch am Beispiel Mensch arbeiten und Schritt für Schritt in Code giessen.
- Definition Objekt (nach Intuitiv, jetzt Formal):
  - Zustand
  - Verhalten
  - Identität
- Terminologie klären: Ich (Instanz) *bin* ein Mensch*(Klasse),* ich *habe* Beine*(Attribute),* ich *kann* laufen *(Methoden)*

- Überleitung: Vorgehen in dieser Reihenfolge

Objekte ...

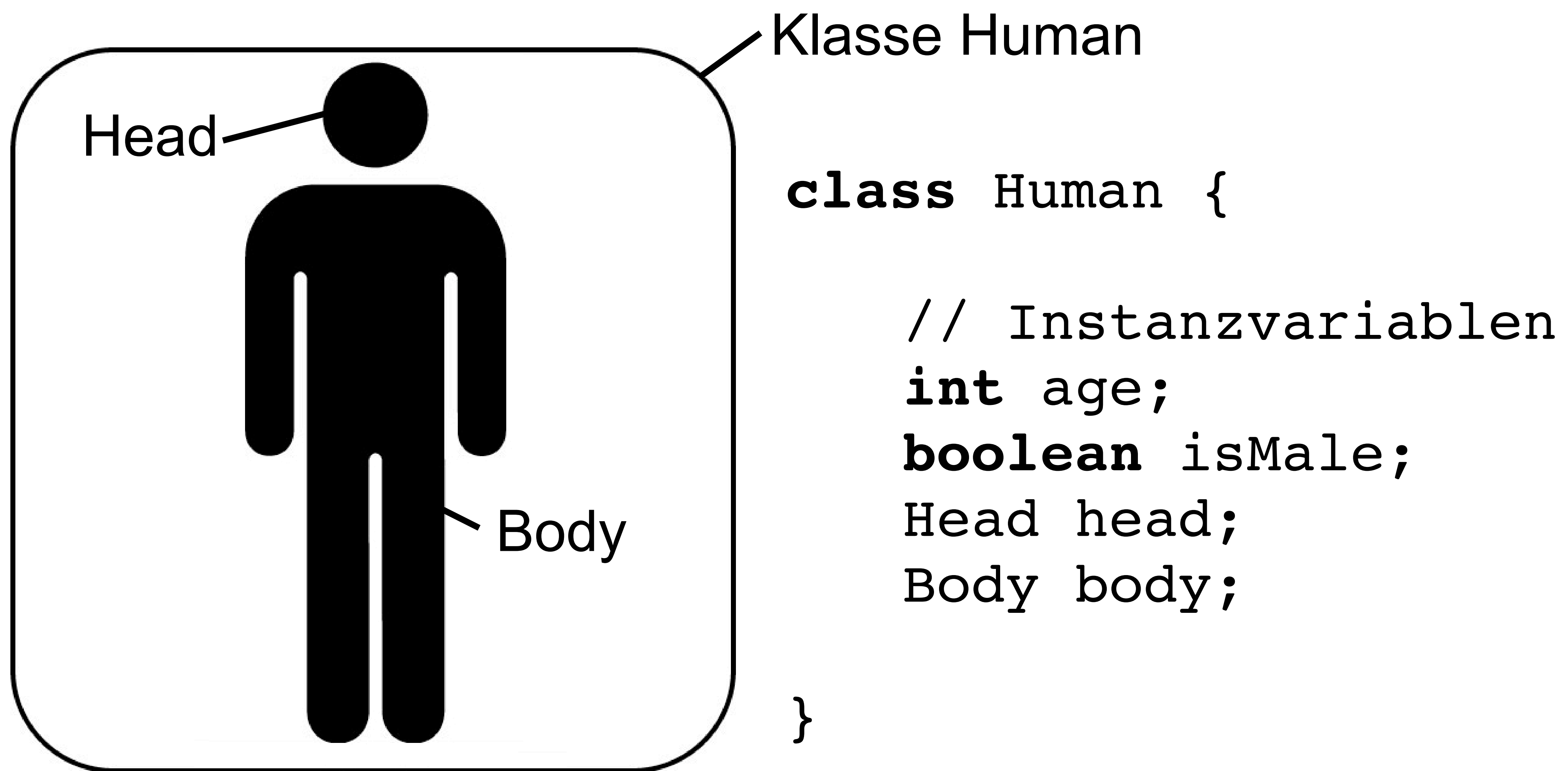
... kapseln einen Zustand
... haben bestimmte Attribute



Speicherung des Zustands in den Attributen eines Objekts.
Zustand kann jederzeit abgefragt und nachträglich verändert werden
Attribute Geschlecht, Alter bestimmen Aussehen eines Menschen.

Klassen ...

... sind Bauvorschriften / Schablonen für Objekte



Klasse schreibt Attribute / Variablen / Eigenschaften
der Instanz / Objekt / Individuum vor

Menschen definiert: Variablen in Klasse
Können primitive Datentypen oder Objekte sein.
Head, Body selbst komplex / Objekt

Namenskonvention: Klasse groß / Variablen klein (Für Lesbarkeit!)
Sprechende Namen: Rolle der Variablen

Einfachster Fall einer Klassendefinition
Kollektion mehrerer Daten unterschiedlicher Typen zu einem neuen Gesamttyp
Abstrakter Datentyp == Klasse in Java

Beispiel ...

```
class Human {
    int age;
    boolean isMale;
    Head head;
    Body body;

    public static void main(String[] args){
        Human katrin= new Human();
        katrin.age= 29;
        katrin.isMale= false;
        katrin.head= new Head();
        katrin.body= new Body();
    }
}
```

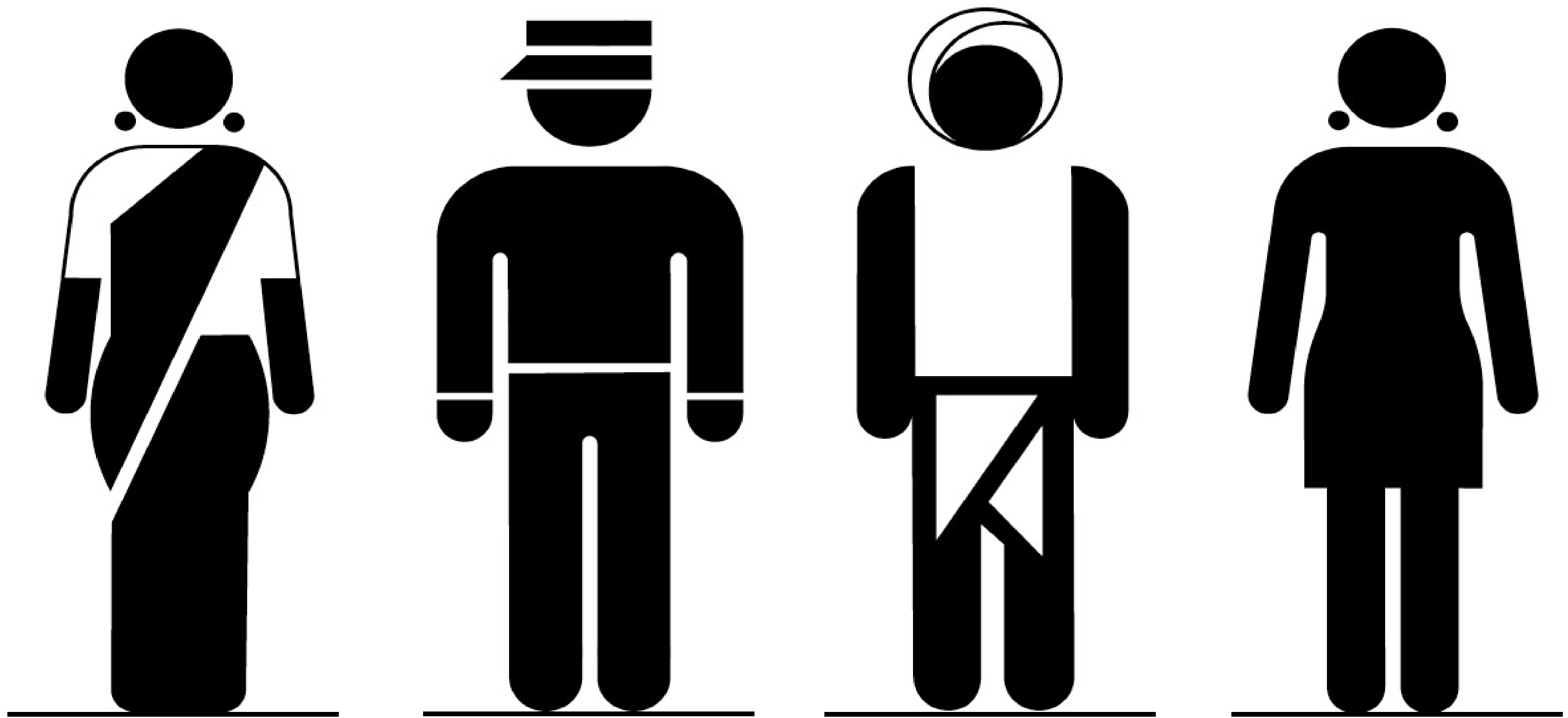
Jede Klasse steht in einer eigenen Datei (eine Klasse pro Datei), die heißen muss wie der Klassenname + .java

new erzeugt eine Instanz der Klasse human, d.h. legt ein neues Objekt an.
-> Erst jetzt enthält die Variable **katrin** das Objekt und Zugriff auf Variablen ist möglich

Punktoperator erlaubt Lese- und Schreibzugriff auf die Instanzvariablen

Objekte einer Klasse ...

... (auch Instanzen, Exemplare) haben ihren eigene Ausfertigung von der Klasse vorgeschriebener Attribute



Fazit: Ihr Kennt jetzt
* Klasse
* Atribute
* Zugriff

Methoden ...

```
class Human {

    int age;
    // ...

    void birthday() {
        this.age++;
    }

    public static void main(String[] args) {
        Human katrin= new Human();
        katrin.age= 29;
        katrin.birthday();
        System.out.println(katrin.age); //>> 30
    }
}
```

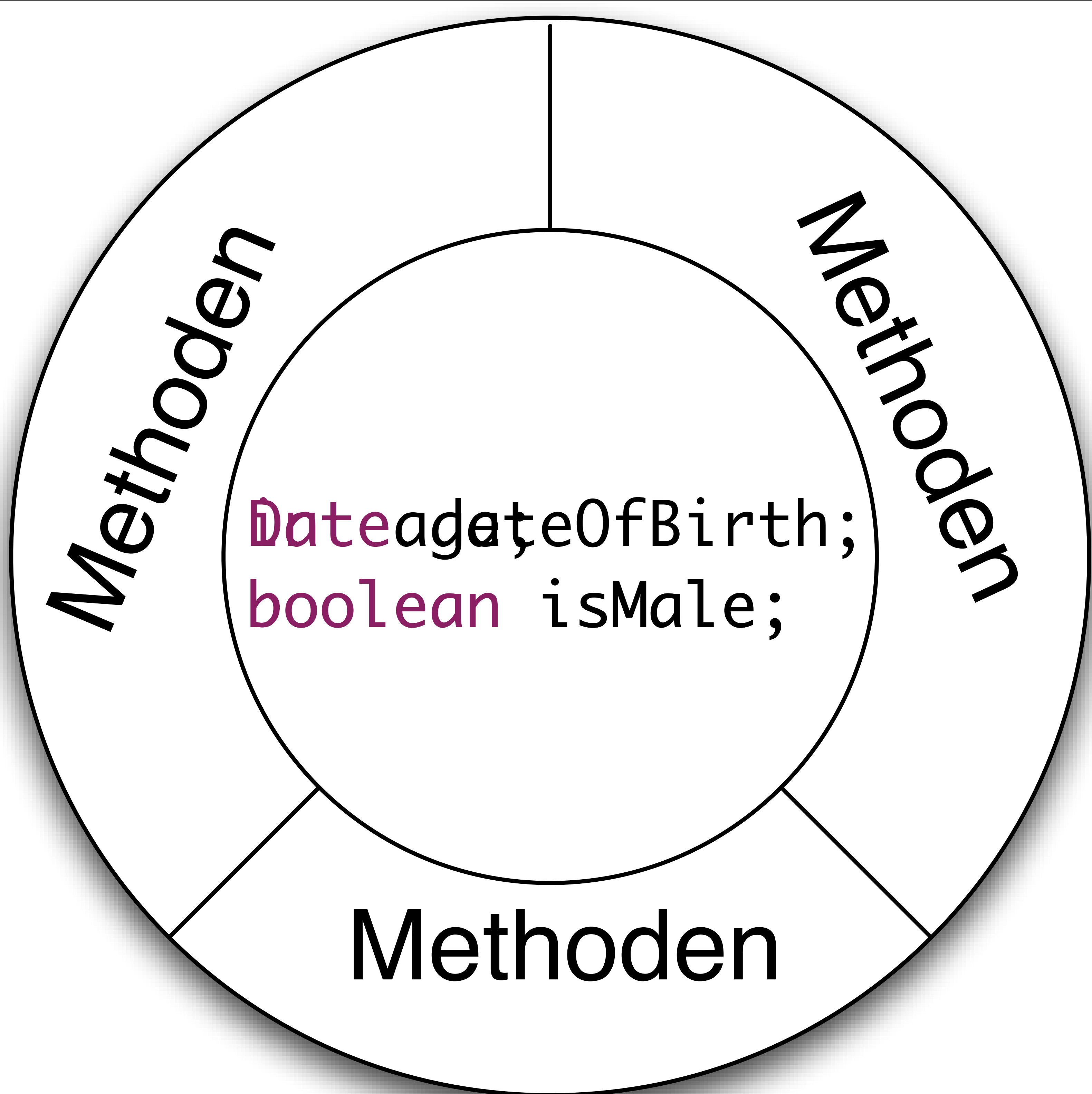
Methoden werden wie die instanzvariablen klein geschrieben. -> Aber Klassen groß

Man beachte, dass draw nicht mehr static ist. Static Methoden beziehen sich nicht auf ein Objekt.

Alle nicht-statischen Methoden benötigen Objekte um aufgerufen werden zu können und haben Zugriff auf die instanzvariablen dieses Objekts.

This bezeichnet die Instanz, auf die ich die Methode gerade aufrufe.

Übergang: Sinn von Methoden



Konzept: Schutzschicht von Methoden (Neue Denke von OO)

Objekt benutzen (age) -> Dann nur über Methoden:
* an keiner stelle im Programm wird direkt age verwendet
* Stattdessen methode birthday()
-> Wie tatsächlich gespeicher: egal

würden austauschen
* **int age; -> Date birthday -> int numberOfWeeks** egal!
* wenn methoden gleich

-> Vorteil: einfacher zu verändern
Andere Sichtweise: Benutzung

Methoden ...

... sind in Java fest einer Klasse zugeordnet.
... Namensgleichheit führt nicht zu Kollision

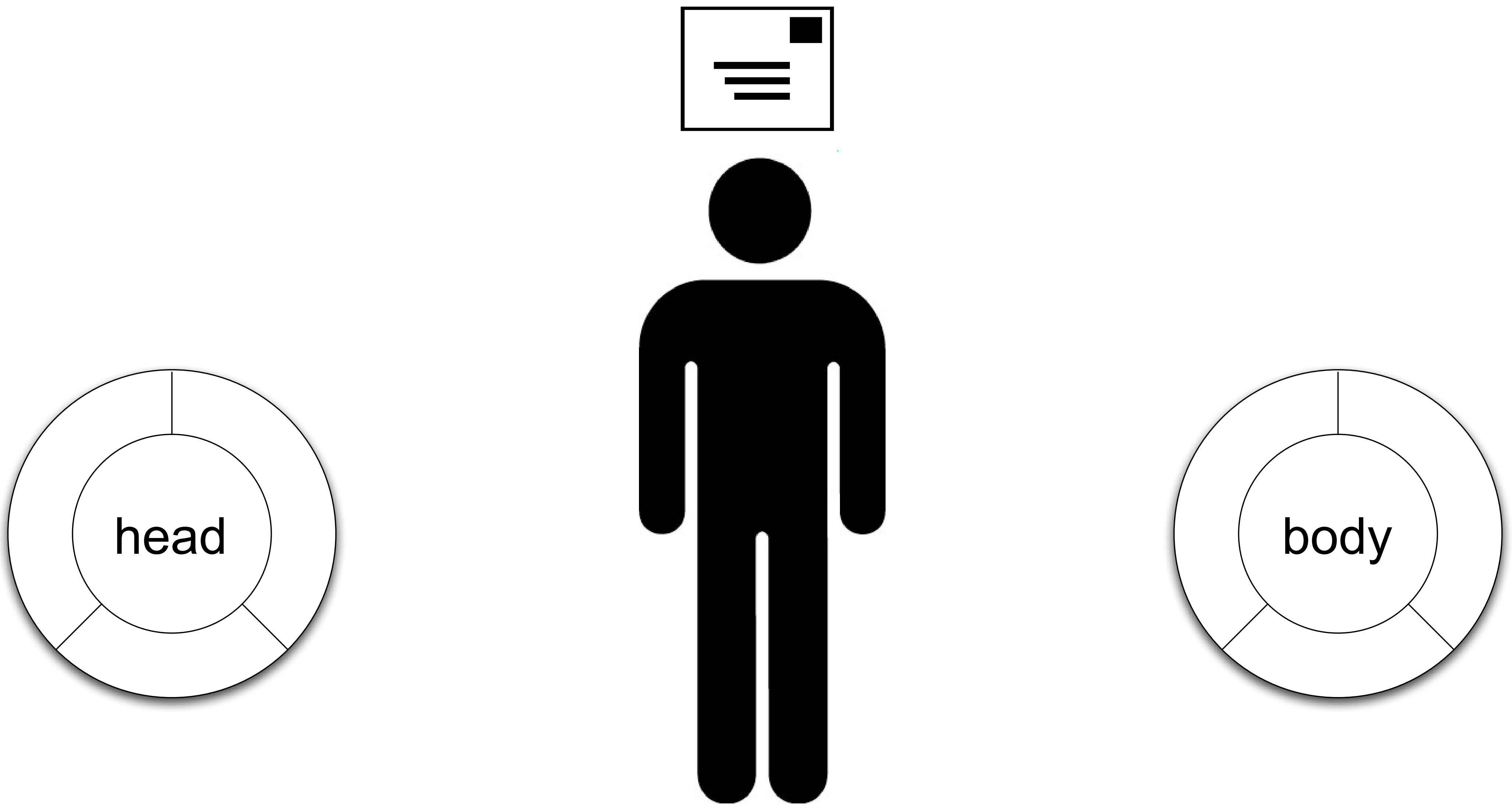
```
class Human {

    int age;
    boolean isMale;
    Head head;
    Body body;

    void draw(){
        this.head.draw();
        this.body.draw();
    }
}
```

Man beachte, dass hier 3 Methoden den gleichen Namen haben, aber etwas unterschiedliches tun.
wenn zwei Klassen Methoden gleichen Namens haben, ist das kein Problem.

Woher weiss ich was jeweils zu tun ist?



Methodenaufruf als Postkarte
Nicht als Brief

Enthält nur Methodennamen und Argumente

head & body erhalten die gleiche Nachricht -> machen aber jeweils was anderes
-> Weil es unterschiedliche Klassen sind.

Überleitung: Problem: Gültige Ins tanzen erstellen

Konstruktoren...

... tragen denselben Namen wie die Klasse

... sind spezielle Methoden, geben neues Objekt zurück

... werden genau einmal aufgerufen

```
Human(){
      this.isMale= false;
      this.age= 0;
      this.head= null;
      this.body= null;
}
```

... werden mit **new** automatisch aufgerufen

```
Human katrin = new Human();
```

Der angegebene konstruktor entspricht dem Standardkonstruktor.

Syntax Erklären: kein Rückgabetyp, kein return, this für zugriff (gezeigt sind Standardwerte)

null: Die Variable ist „leer“ / noch kein Objekt vorhanden

Wird er nicht explizit angegeben, erzeugt Java ihn automatisch.

Der Standardkonstruktor ist immer das erste, was bei der erzeugung eines neuen objekts aufgerufen wird.

Er initialisiert die instanzvariablen in der angegebenen Weise. (standardwerte erklären)

Aus Dokumentationsgründen ist es immer besser, Konstruktoren explizit anzugeben.

Konstruktoren...

... können Initialisierung von Instanzvariablen erzwingen

... können überlagert werden

```
Human(boolean isMale){

    this.isMale= isMale;
    this.age= 0;
    this.head= new Head();
    this.body= new Body();

}
```

... werden mit **new** automatisch aufgerufen

```
Human katrin = new Human(false);
```

Wird ein Konstruktor mit Argumenten angegeben, so ist jeder Benutzer der Klasse gezwungen, diesen auch zu benutzen.

Der Standardkonstruktor ist nach aussen hin dann nicht mehr verfügbar, wird aber intern trotzdem aufgerufen, so dass die initialisierung gewährleistet wird.

Konstruktoren können überladen werden! (C-tor chaining wenn Zeit ist)

Fazit: Syntax. Rückgabetyp, static, return, this, argumente

Überleitung: Was kann man falsch machen?

Vorsicht...

... wo liegt hier der Fehler?

```
Human(boolean isMale){

    isMale= isMale;
    int age= 0;
    Head head= null;
    Body body= null;

}
```

BAAAAD

```
Human(boolean isMale){

    this.isMale= isMale;
    this.age= 0;
    this.head= new Head();
    body= new Body();

}
```

GOOOOD

Frage - PAUSE!!!!

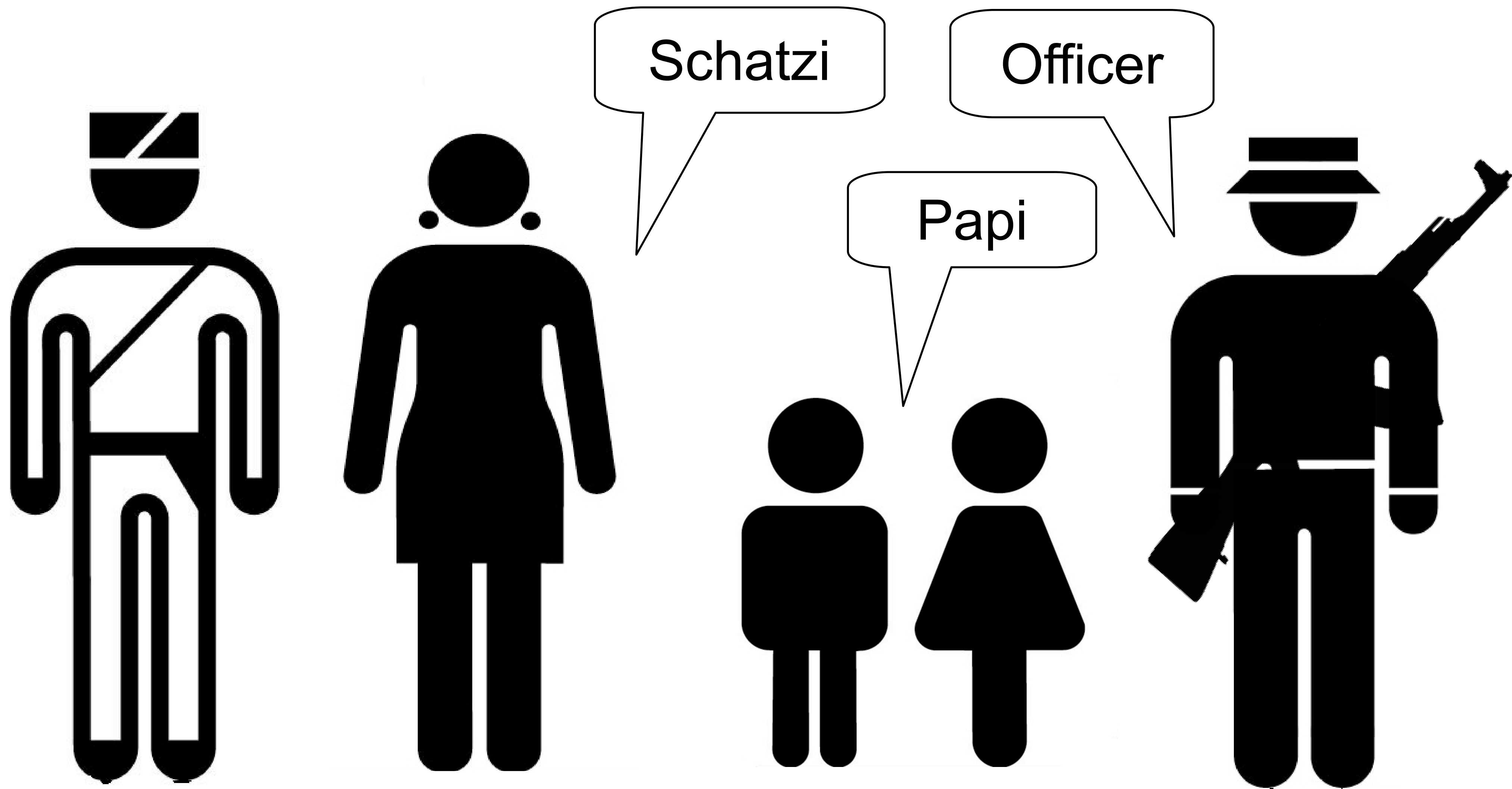
Probleme:
Scope: isMale (instanzvariable) ist durch isMale (argument) „verschattet“ / „hidden“
* this verwenden! Wir empfehlen für den Anfang immer this zu verwenden
** Später können immer noch darauf verzichten.
* Neue Variablen verschatten Instanzvariablen
** this...

Konstruktor Variablen überschreiben
this, wann brauchts das? (Scope)

Überleitung: was steht genau in Variablen?

Referenzen...

... repräsentieren ein Objekt.
Eine Variable speichert diese Referenz.



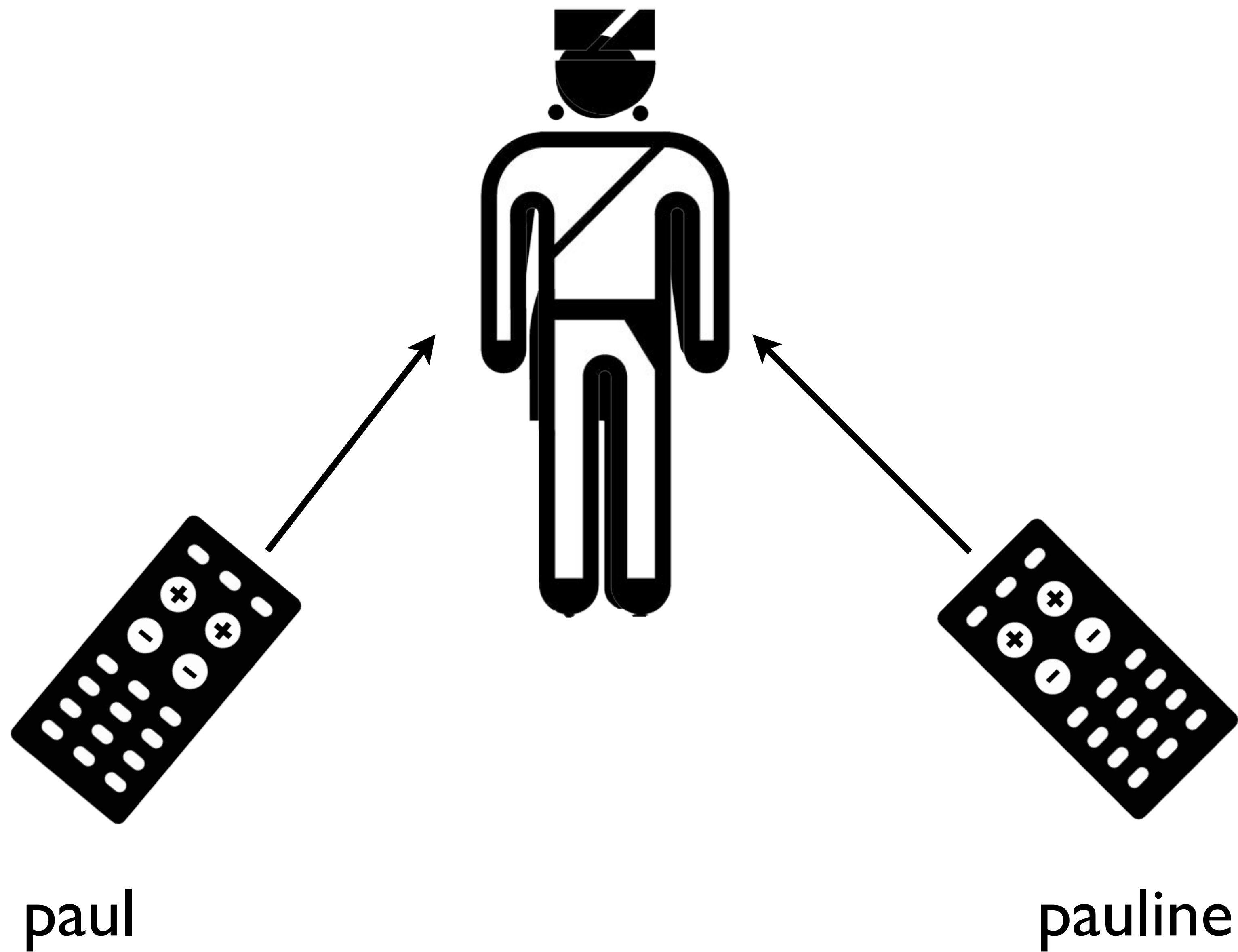
Neues Konzept: Viele Namen für das gleiche „Paul“
Frau: Variable „Schatzi“
Kinder: Variable „Papi“
Verbrecher: Variable „Officer“

Referenzen ...... repräsentieren ein Objekt, und eine Variable speichert diese Referenz.
Objektvariablen speichern Referenzen (Pointer)

So sind in normaler Welt gewohnt. -> Programmierung: Verweis

Für Programmierer:
Überleitung: Veränderung über einen Namen....
Geheimes Leben: Nachtclub: Pauline

Referenzen...

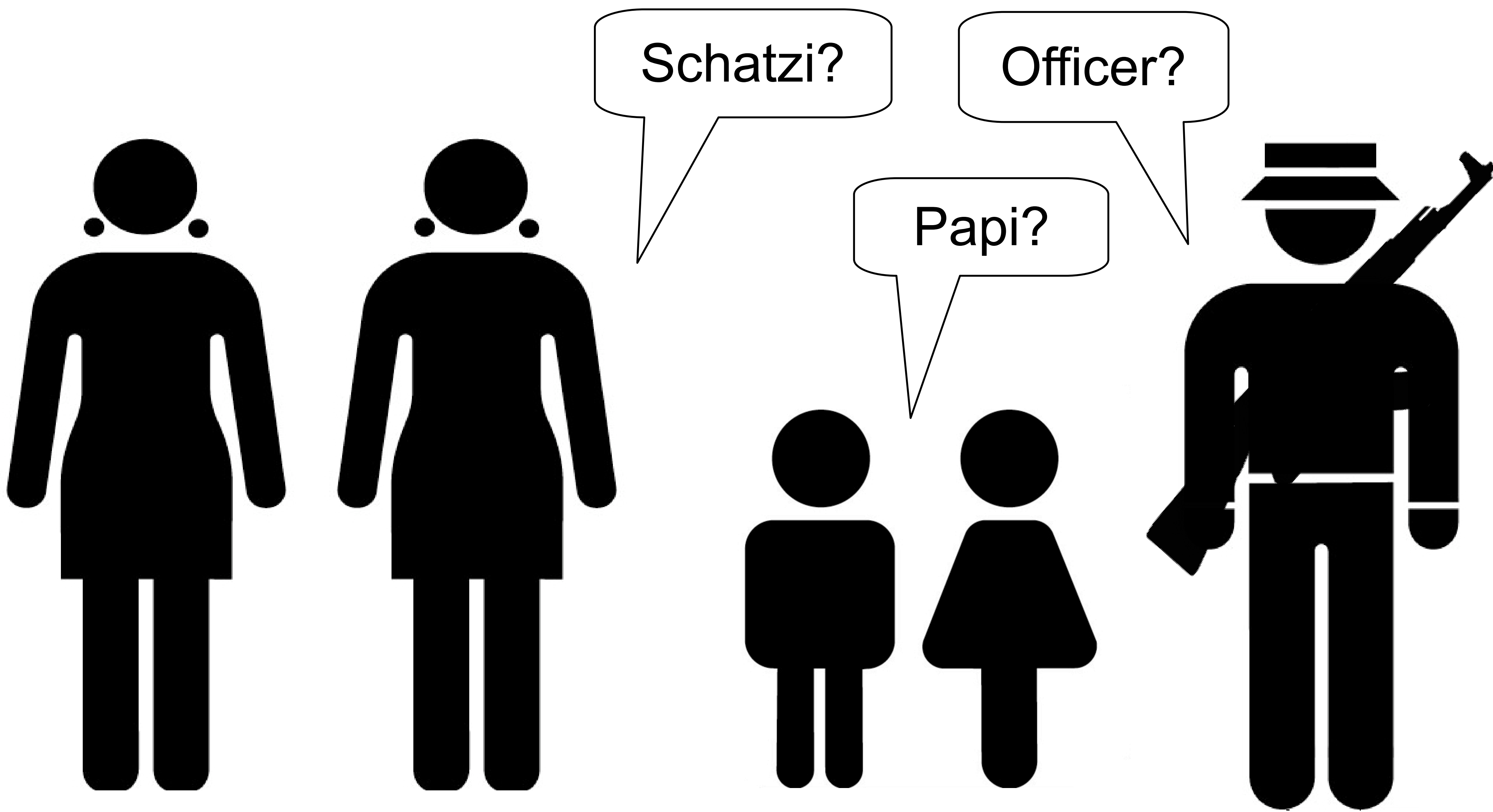


Geheimes Nachtleben von Paul....

Referenzen sind „Fernbedienungen“ -> Wir können Methoden aufrufen

Geschlechtsänderung! -> Alle Referenzen zeigen auf Frau!

Primitive vs. komplexe Datentypen



Mit allen Konsequenzen....

-> Darum wurde funktionale Programmierung erfunden....

Primitive vs. komplexe Datentypen

```
class Hospital {
   void changeGender(boolean gender) {
      System.out.println(gender); // true
      gender = ! gender;
      System.out.println(gender); // false
   }
   static void main(String [] ignored) {
      Human paul = new Human(true);
      boolean paulIsMale = paul.isMale; // true
      Hospital butchers = new Hospital();
      butchers.changeGender(paul.isMale);
      System.out.println(paul.isMale); // true
      System.out.println(paulIsMale); // true
   }
}
```

Wichtiger Unterschied.
Daher: Paulines Lebensgeschichte in Code.
Einmal mit primitiven Datentypen
Einmal mit Objekten / Referenzen

Beste methode zum lernen: Solche Test-Methoden schreiben!

code erklären!

Überleitung: jetzt richtig

Primitive vs. komplexe Datentypen

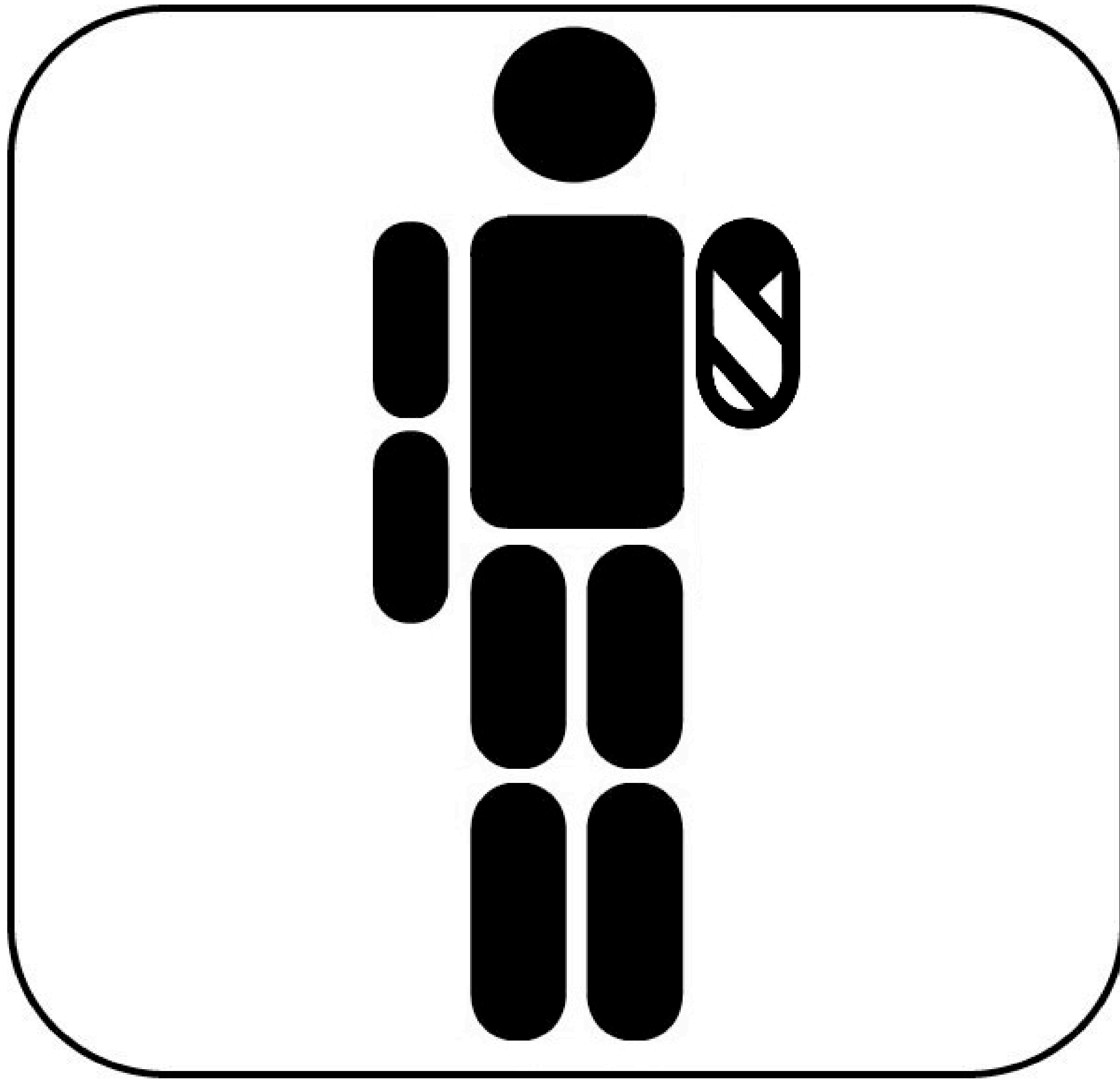
```
class Hospital {
    void changeGender(Human who) {
        System.out.println(who.isMale); // true
        who.isMale ^= true;
        System.out.println(who.isMale); // false
    }
    static void main(String [] ignored) {
        Human paul = new Human(true);
        Human pauline = paul;
        Hospital butchers = new Hospital();
        butchers.changeGender(pauline);
        System.out.println(pauline.isMale); // false
        System.out.println(paul.isMale); // false
    }
}
```

code erklären!
Fernbedienung, neuer Name, teilen sich Objekt -> Referenz
Das ist anders!

Wenn man sie also kopiert -> dann hat man nur zwei „Fernbedienungen“, nicht zwei Datenobjekte
-> genauso bei arrays!
-> alles was mit „new“ angelegt wird

DAS unterscheidet diese Daten von int, double, boolean
Da hat man wirklich kopien gemacht
(Historisch: wegen Performance, wäre heute unwichtig / anders gelöst)

Nullreferenzen



... kann Datenstrukturen terminieren (eher vermeiden)
... vermeiden uninitialisierte Referenzen

Häufiger Fehler: NullPointerException
Wegen:
* Initialisierung Vergessen
* null gesetzt

Überleitung: Nochmal in Code

Initialisierung von Variablen

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

Compilerfehler!!!

```
Human nobody = null;
nobody.birthday();
```

Laufzeitfehler!!!

```
Human somebody = new Human();
somebody.birthday();
Human alsoSomebody = somebody;
alsoSomebody.birthday();
```

... terminieren Datenstrukturen

...vermeiden uninitialisierte Referenzen

Daher Häufiger Fehler: NullPointerException

Nach solchen Fehlern suchen!

Empfehlung: null möglichst vermeiden / nicht benutzen. Null generell (nach möglichkeit) nicht als gültiges Argument beim Programmieren verwenden

```java
class Human {
  int age;
  Head head;

  Human(int anAge) {
    this.age = anAge;
    head = new Head();
  }

  void speak(String what) {
    head.shoutOutLoudly(what);
  }

  static void main(String [] args) {
    Human martin = new Human(27);
    martin.speak("Buh!");
  }
}
```

Eine Klasse pro Datei
Klasse kann enthalten: Variablen (Primitiv, Komplex), Methoden (Normal / Konstruktor)
Komplex: Referenzen, anderes Verhalten als Primitive Datentypen
Methoden:
* Konstruktor: kein Rückgabetyp, kein return
* Normale: ohne static!
Klasse, Type Großgeschrieben
Variablen / Methoden klein
Spezialfall Konstruktor
Zugriff auf Instanzvariablen -> this
Verschattung durch Argumente -> this, wenn eindeutig, kann weglassen
* Anfangs lieber zu oft benutzen
* Namenskonvention vermeidet Kollisionen: this.age (instanzvariable) anAge (argument)

Fazit: *Klassen* bestimmen *Attribute* und *Methoden* ihrer *Instanzen*

Jetzt könnt Ihr OO-Programmieren. Fragen dazu?
Übergang: Themawechsel: Gutes OO Design

OO-Design

Cheap, Fast, Right. Pick two.

Objektorientiertes Design

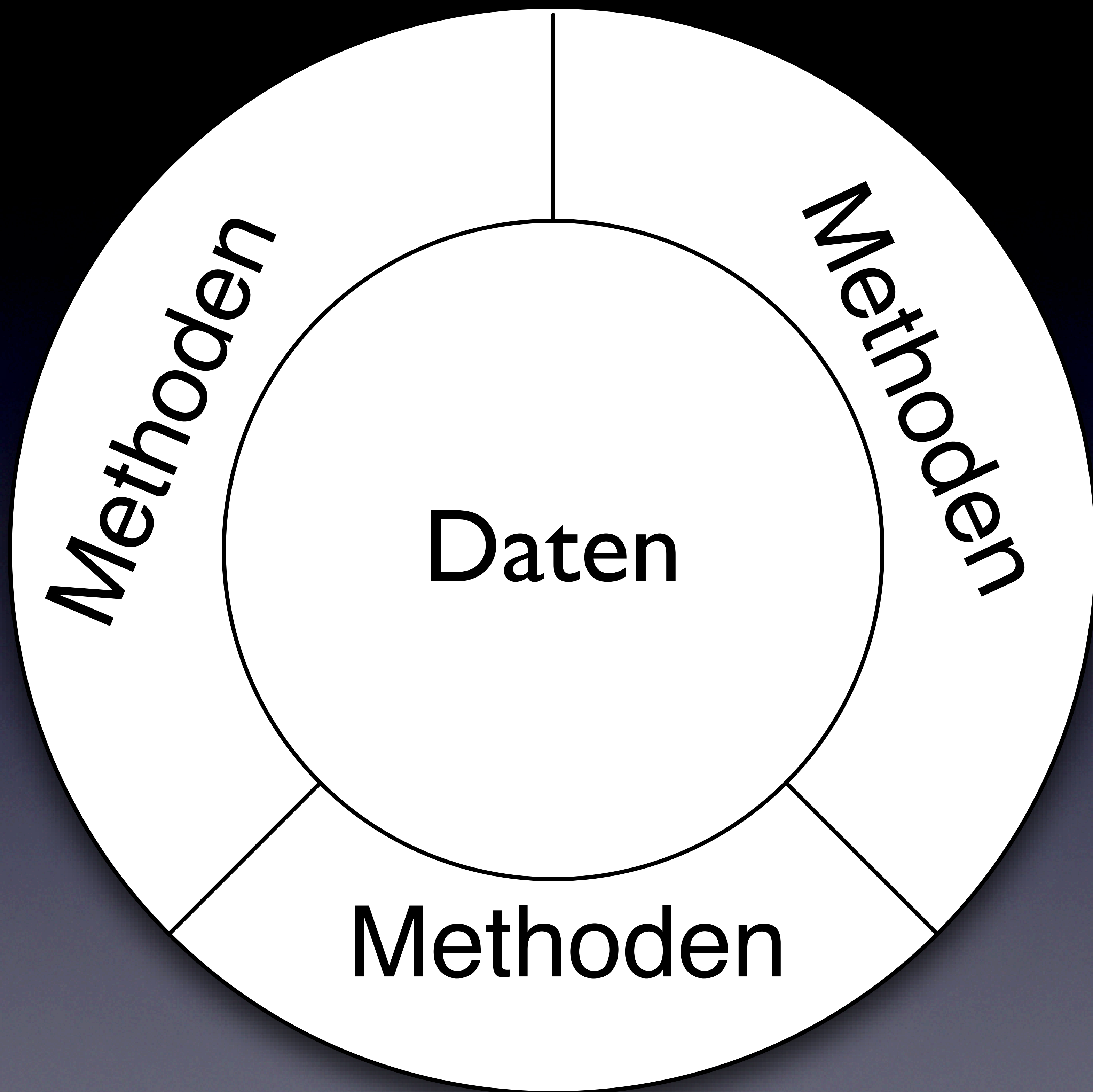
Der Sinn von Design:
* Faulheit -> Duplikation von Code vermeiden.
* Wichtigstes Ziel: Code gut Lesbar / Verständlich

! Alles sind Abwägungen: Engineering Triangle: „Cheap, Fast, Right: Pick two!“

Advanced! -> Jetzt hören, Später umsetzen
Duplikation auf vielen Ebenen
* Code -> Copy‘n‘Paste -> „Manuelle Vererbung“
* Abläufe -> Ähnlicher Code
* Konzepte -> Ständig Schleifen, Ständig Array
* Meta-Ebenen -> Ähnliche Objekte

-> Sinn -> Duplikation vermeiden -> Weniger Code

Überleitung: Anderer Sinn: Abhängigkeiten minimieren (DANACH!)



Wie macht mans:
* Methoden als Schutzschild
* Variablen nur innerhalb ansprechen

Das ist das Mantra für OO-Code
* (Later) Wohin gehört Code? In Das objekt dessen Daten verwendet werden!

Überleitung: Wie kommt man auf Objekte?

Stärke von OO! -> Simulation (der Wirklichkeit)

Objekte Finden:
* Von Konkret (n00b) nach Abstrakt (l33t)

Konkrete: Schachbrett, Figuren
Abstrakt: Regeln (Austauschen für neues Spiel)

Das ist alles was es zu Objektorientiertheit zu sagen gibt
Jetzt könnt ihr OO-Programmieren

! „Easy to learn, hard to master“

JUNGE PROGRAMMIERER: Gehe nach Afrika;   Beginne am Kap der guten Hoffnung;   Durchkreuze Afrika von Süden nach Norden bidirektional in Ost-West-Richung;
Für jedes Durchkreuzen tue:
    Fange jedes Tier, das Du siehst;   Vergl. jedes gefangene Tier mit einem Elefanten bekannten Tier;
    Halte an bei übereinstimmung;

ERFAHRENE PROGRAMMIERER: verändern Algorithmus A, indem sie ein als Elefant bekanntes Tier in Kairo plazieren, damit das Programm in jedem Fall korrekt beendet wird.

LISP-PROGRAMMIERER: bauen einen Irrgarten aus Klammern und hoffen, daß sich der Elefant darin verirrt.

EXTREME PROGRAMMER: bestehen darauf, dass erst einmal ein Test für Elefanten vorhanden sein muss, bevor wir versuchen, sie zu fangen!

OO-PROGRAMMIERER: bestehen darauf, daß der Elefant eine Klasse sei, somit schließlich seine Fang-Methoden selbst mitzubringen habe.

schwer: zu welchem Objekt soll Methode?
* fanden wir auch schwer
* praxis, praxis, praxis

Viel spaß in der Übung, wo ihr diese Konzepte jetzt umsetzen werdet