

# **EINI**

# **LogWing/WiMa/MP**

**Einführung in die Informatik für  
Naturwissenschaftler und Ingenieure**

**Vorlesung      2 SWS      WS 22/23**

**Dr. Lars Hildebrand**  
**Fakultät für Informatik – Technische Universität Dortmund**  
**[lars.hildebrand@tu-dortmund.de](mailto:lars.hildebrand@tu-dortmund.de)**  
**<http://ls14-www.cs.tu-dortmund.de>**

## ▶ Kapitel 6

### Objektorientierte Programmierung – Einführung

EINI LogWing /  
WiMa

#### Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

## ▶ Unterlagen

- ▶ Dißmann, Stefan und Ernst-Erich Doberkat: *Einführung in die objektorientierte Programmierung mit Java*, 2. Auflage. München [u.a.]: Oldenbourg, 2002.  
(→ ZB oder Volltext aus Uninetz)
- ▶ Echtele, Klaus und Michael Goedicke: *Lehrbuch der Programmierung mit Java*. Heidelberg: dpunkt-Verl, 2000.  
(→ ZB)

In diesem Kapitel:

- Prolog
- Einführung

# Übersicht

## Begriffe

- ✓ Spezifikationen, Algorithmen, formale Sprachen
- ✓ Programmiersprachenkonzepte
- ✓ Grundlagen der imperativen Programmierung
  
- ✓ Algorithmen und Datenstrukturen
  - ✓ Felder
  - ✓ Sortieren
  - ✓ Rekursive Datenstrukturen (Baum, binärer Baum, Heap)
  - ✓ Heapsort
  
- Objektorientierung
  - Einführung
  - ▶ Vererbung
  - ▶ Anwendung

EINI LogWing /  
WiMa

### Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

In diesem Kapitel:

- Prolog
- Einführung

# Gliederung

- ▶ Elemente höherer Programmiersprachen
- ▶ Warum Klassen und Objekte?
- ▶ Aufbau eines Java-Programms
  - ▶ Syntaktische Struktur
  - ▶ Syntaxdiagramme
  - ▶ Attribute
  - ▶ Methoden
  - ▶ Instanziierung
- ▶ Details der objektorientierten Programmierung in Java

EINI LogWing /  
WiMa

## Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

### In diesem Kapitel:

- Prolog
- **Einführung**

## Was haben wir bis jetzt?

- ▶ Einfache **Anweisungen**
- ▶ Zusammenfassung von mehreren Anweisungen zu einer **Funktion** (oder Unterprogramm oder Prozedur, ...)
- ▶ Einfache **Datentypen**
  - ▶ Einzelne Werte werden selten betrachtet  
→ daher: ganze Wertemengen
  - ▶ Grundvorrat an einfachen (primitiven) Wertemengen ist durch Java vorgegeben:
    - ein Abschnitt der ganzen Zahlen
    - eine Teilmenge der rationalen Zahlen
    - einfache Zeichen und Zeichenketten
    - Wahrheitswerte (wahr, falsch)

# Elemente höherer Programmiersprachen II

- ▶ Die üblichen Operationen auf diesen Mengen werden durch Java zur Verfügung gestellt:

→ **Wertemengen + Operationen = Datentyp**

- ▶ Primitive Wertemengen werden als Basis für **komplexe Strukturen** benutzt.
- ▶ Vorschriften, wie einzelne einfache Daten zu komplexen Gebilden geformt werden

→ **komplexe Datenstrukturen:**

- ▶ Feld/Array

- ▶ ...

EINI LogWing /  
WiMa

## Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

In diesem Kapitel:

- Prolog
- **Einführung**

# Warum Klassen und Objekte?

Die Trennung von Verfahrensvorschrift (Algorithmus) und Datenstrukturen hat sich als hinderlich erwiesen.

- ▶ Besonders wichtig für das Verständnis von Programmen ist die **Zusammenfassung** der zulässigen Verfahren und der dazugehörigen Datenstrukturen.
- ▶ Damit können auf einzelne Objekte unseres Problems zugeschnittene Verfahren formuliert werden, wie z. B.:
  - ▶ Wie wird dieses Objekt erzeugt?
  - ▶ Wie wird es ausgegeben?
  - ▶ ...
- ▶ Diese Zusammenfassung von Objektdaten und Verfahren findet in einer **Klasse** statt.

EINI LogWing /  
WiMa

## Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

In diesem Kapitel:

- Prolog
- **Einführung**

# Klassen von Objekten

In der Regel ist es mühsam, alle betrachteten Objekte einzeln zu beschreiben ...

- ▶ Man ist nicht an einer einzelnen Person, sondern an allen Personen interessiert.
  
- ▶ Daher:
  - ▶ Zusammenfassung aller **gleichartigen Objekte** zu einer **Klasse von Objekten**
  - ▶ In den Programmen stehen dann nur:
    - Definitionen von Klassen
    - Verfahren, wie einzelne Exemplare (Objekte, Instanzen) geschaffen werden können.



# Was macht Objektorientierung aus?

- ▶ **Objekte:**
  - ▶ Vereinigung von Algorithmus und Datenstrukturen
- ▶ Verallgemeinerung und Abstraktion:
  - ▶ **Klassen** von Objekten
- ▶ **Neue Begriffe** in diesem Umfeld:
  - ▶ Objekt
  - ▶ Instanz
  - ▶ Klasse
  - ▶ Methode
  - ▶ Vererbung
  - ▶ Interface
  - ▶ ...

EINI LogWing /  
WiMa

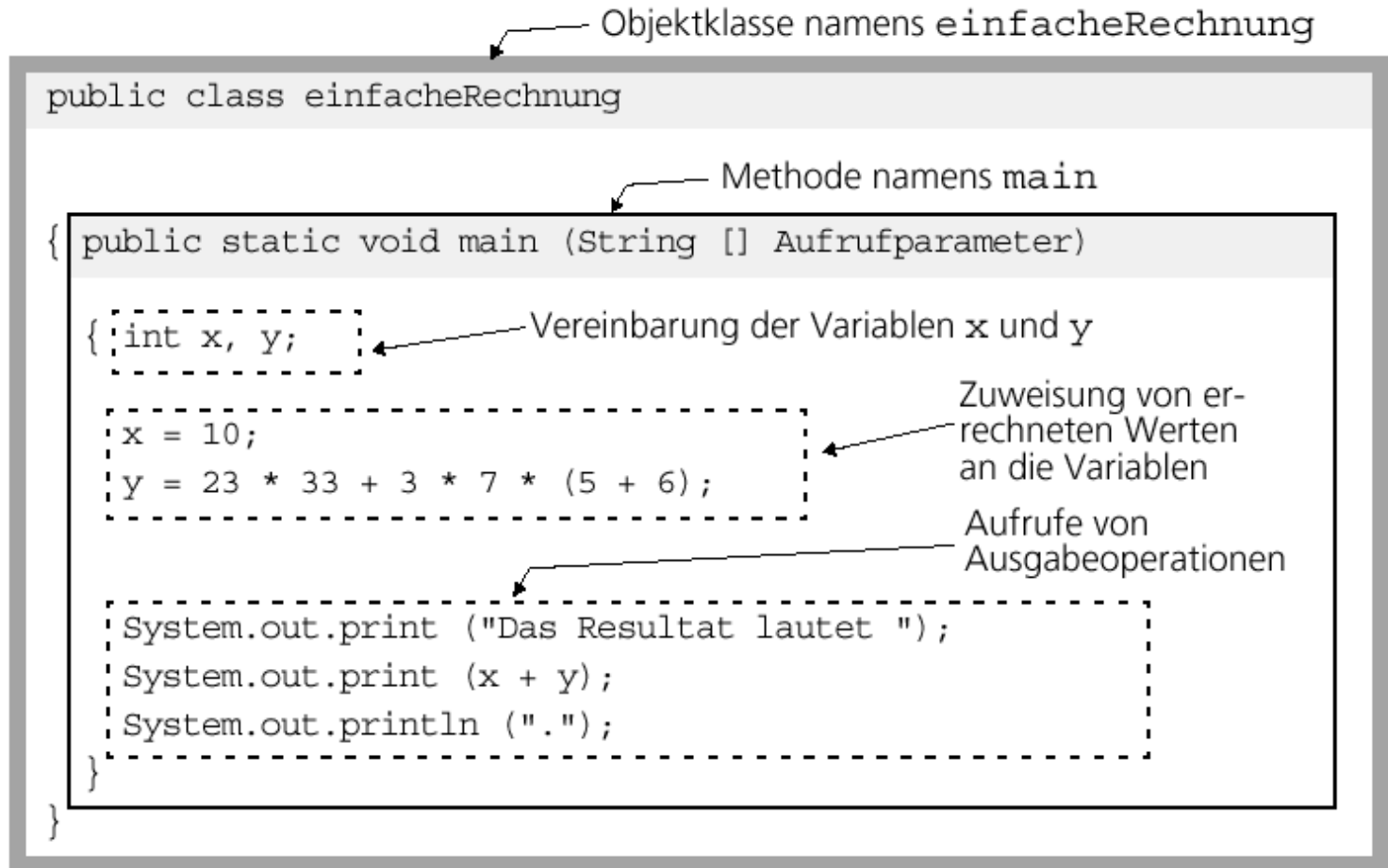
## Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

### In diesem Kapitel:

- Prolog
- **Einführung**

# Aufbau eines Java-Programms I



Echtle/Goedicke, Heidelberg: *Prog. 2-2*, S. 22 © dpunkt 2000.

EINI LogWing /  
WiMa

## Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

In diesem Kapitel:

- Prolog
- **Einführung**

# Aufbau eines Java-Programms II

## Syntaktische Struktur von (Java-)Programmen

- ▶ Die syntaktische Struktur von (Java-)Programmen ist durch Schachteln von **Blöcken** gegeben.
- ▶ Generell bestehen solche Schachteln aus einem **Kopf** und einem **Rumpf**:

```
public class einfacheRechnung {  
    ...  
    public static void main( ... ) {  
        ...  
    }  
}
```

# Aufbau eines Java-Programms III

- ▶ Java-Programm:  
Folge von **Klassendefinitionen**, die wiederum u.a. eine Folge von **Methodendefinitionen** enthalten.
- ▶ Im Beispiel (vorherige Folie) gab es:
  - ▶ Die Definition einer Klasse **einfacheRechnung**
  - ▶ Innerhalb dieser Klasse die Definition einer Methode **main**
- ❖ **main** spielt eine besondere Rolle, da sie die Methode ist, die automatisch als erste aufgerufen wird, wenn man das Programm startet.

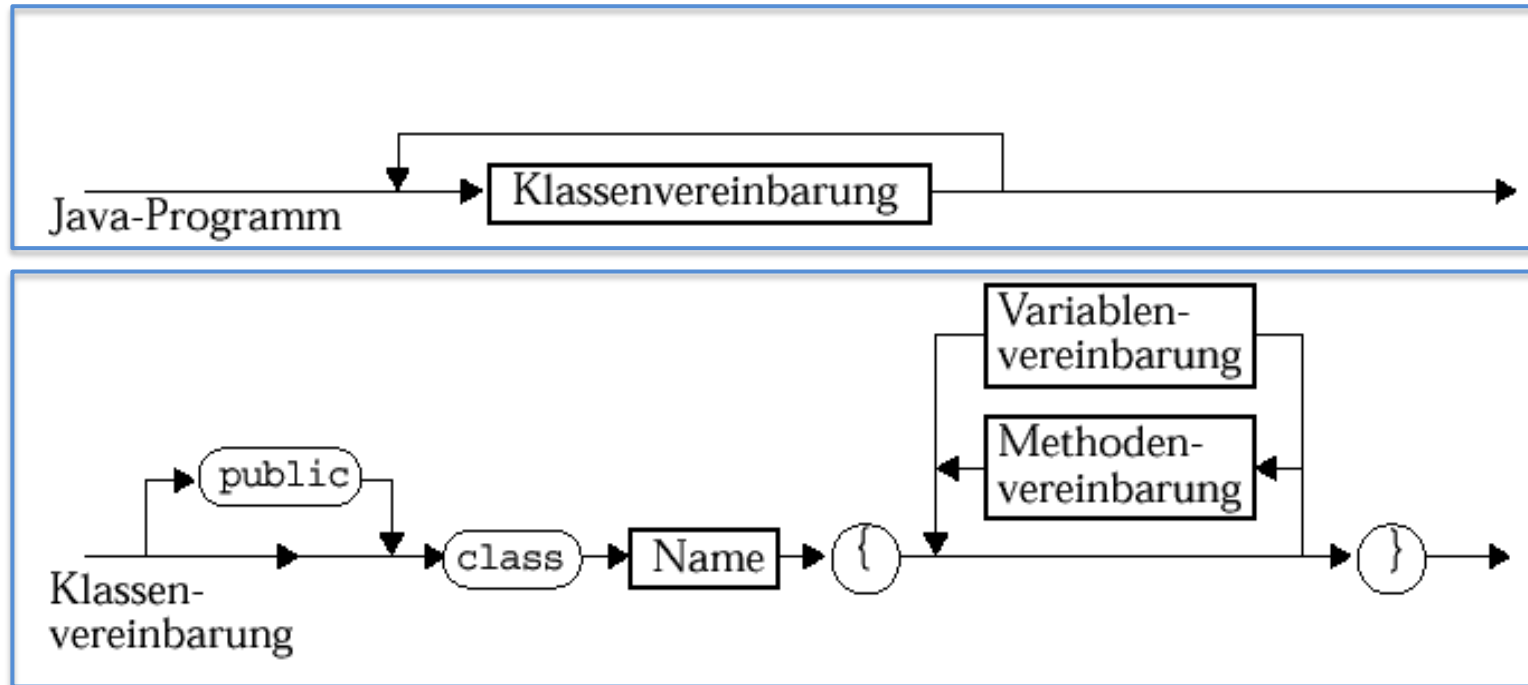
# Aufbau eines Java-Programms IV

## Syntaxdiagramme für einfache Java-Programme (1)

EINI LogWing /  
WiMa

### Kapitel 6

Objektorientierte  
Programmierung -  
Einführung



Echtle/Goedicke, Heidelberg: *Abb. 2-2 (Ausschnitt)*, S. 29 © dpunkt 2000.

### In diesem Kapitel:

- Prolog
- **Einführung**

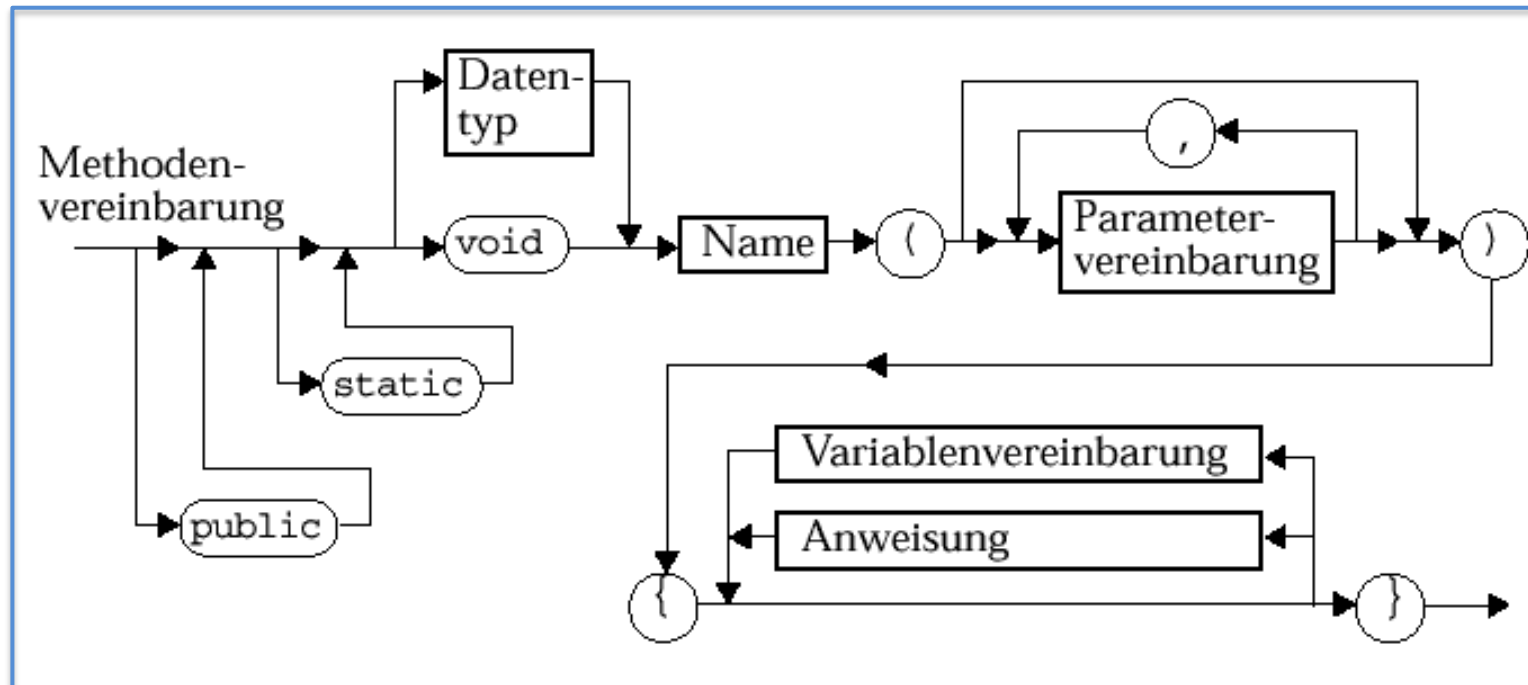
# Aufbau eines Java-Programms V

## Syntaxdiagramme für einfache Java-Programme (2)

EINI LogWing /  
WiMa

### Kapitel 6

Objektorientierte  
Programmierung -  
Einführung



Echtle/Goedicke, Heidelberg: *Abb. 2-2 (Ausschnitt)*, S. 29 © dpunkt 2000.

### In diesem Kapitel:

- Prolog
- **Einführung**

# Aufbau eines Java-Programms VI

```
01 public class Girokonto {
02     private int kontostand;    // in Cent gerechnet
03
04
05     public Girokonto() {
06         kontostand = 0;
07     }
08
09     /* Einzahlen und Abheben */
10     public void zahle (int cent){
11         kontostand = kontostand + cent;
12     }
13
14     public int holeKontostand() {
15         return (kontostand);
16     }
17 }
```

EINI LogWing /  
WiMa

## Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

In diesem Kapitel:

- Prolog
- Einführung

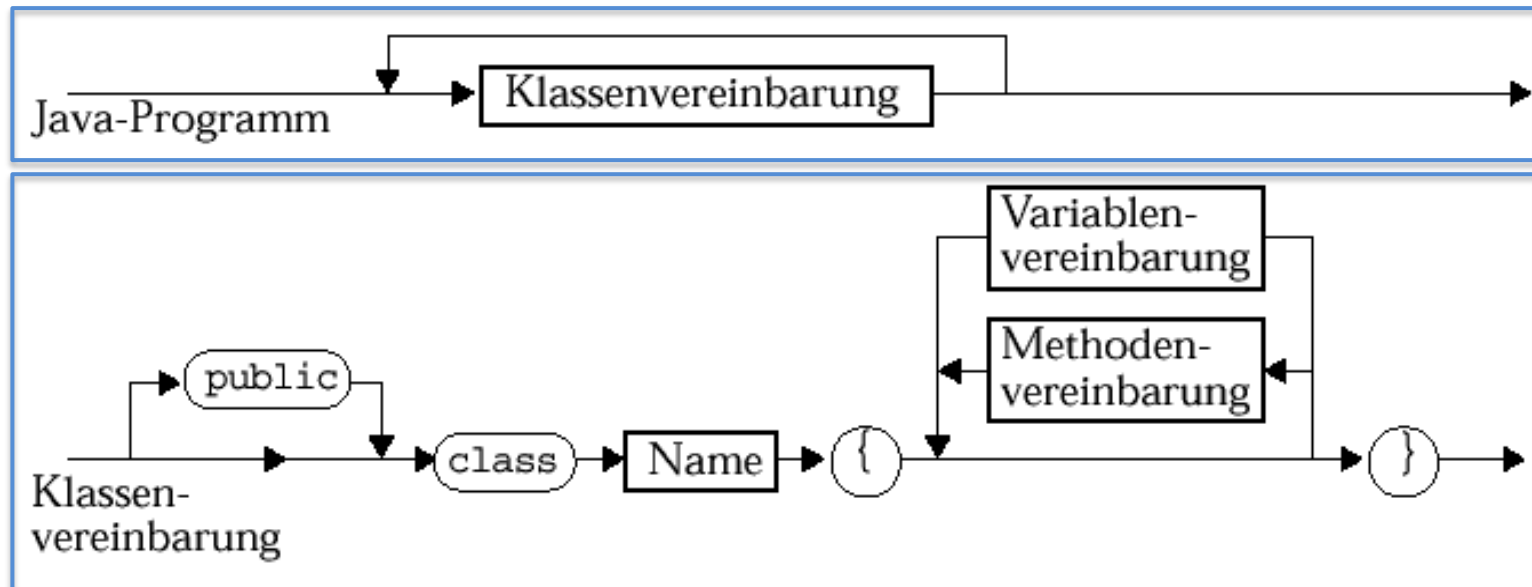
# Aufbau eines Java-Programms VII

```
public class Girokonto {  
    private int kontostand;  
  
    public Girokonto() {  
        kontostand = 0;  
    }  
    ...  
}
```

EINI LogWing /  
WiMa

## Kapitel 6

Objektorientierte  
Programmierung -  
Einführung



Echtle/Goedicke, Heidelberg: *Abb. 2-2 (Ausschnitt)*, S. 29 © dpunkt 2000.

In diesem Kapitel:

- Prolog
- **Einführung**



# Attribute: Teile des Gesamtzustands eines Objekts

Beispiel:

```
private int kontostand;
```

```
private Girokonto meinKonto;
```

```
Modifier Datentyp attributname
```

▶ Verändern von Attributen:

```
kontostand = kontostand + cent;
```

▶ Kapseln von Attributen (→ Ziel: möglichst alle kapseln)

- ▶ Privat (**private**): nur innerhalb des Objekts sichtbar
- ▶ Öffentlich (**public**): auch außerhalb des Objekts sichtbar
- ▶ Zugriffe auf gekapselte Attribute nur durch einzelne Methoden des Objektes (**getter & setter**)

# Methoden I: Deklaration/Definition

## Teile des Verhaltens eines Objekts:

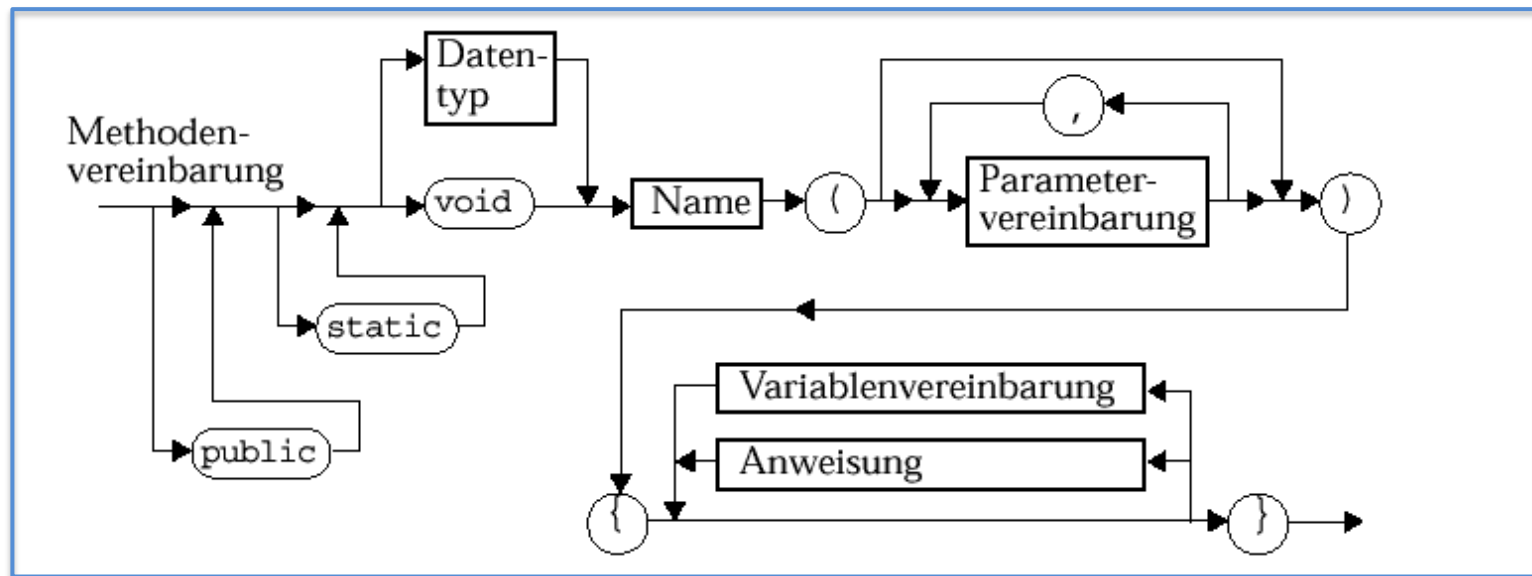
```
public void zahle (int cent){  
    kontostand = kontostand + cent;  
}
```

```
public int holeKontostand() {  
    return (kontostand);  
}
```

EINI LogWing /  
WiMa

### Kapitel 6

Objektorientierte  
Programmierung -  
Einführung



Echtle/Goedicke, Heidelberg: Abb. 2-2 (Ausschnitt), S. 29 © dpunkt 2000.

### In diesem Kapitel:

- Prolog
- Einführung

# Methoden II: Aufruf

- ▶ Aufruf von Methoden eines Objektes:
  - ▶ Punkt-Notation, qualifizierter Zugriff
  - ▶ `int betrag = meinKonto.holeKontostand();`
  - ▶ `meinKonto.zahle(100);`
- ▶ Kapseln von Methoden
  - ▶ **private**: nur als interne Hilfsmethode verwendbar
  - ▶ **public**: öffentlich, d.h. extern sichtbar

# Spezielle Methode "main"

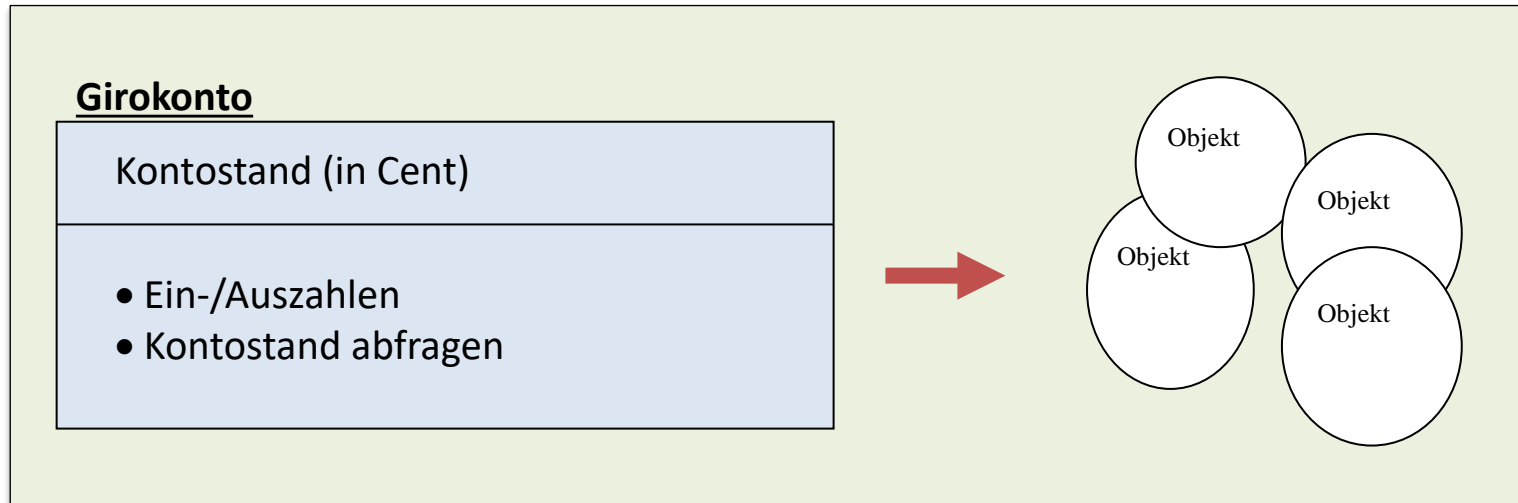
- ▶ **Einsprungspunkt** bei "Ausführen" einer Applikation
  - ▶ Übergeben von Argumenten aus der Kommandozeile möglich
  - ▶ für Testen von Klassen-Implementierungen geeignet

```
public class TestGirokonto {  
    ...  
  
    public static void main(String[] args) {  
        ... // Hier Code zum Testen einfügen  
    }  
  
    ...  
}
```

# Instanziierung

Wir haben nun die Klasse.

## Wie entstehen Objekte?



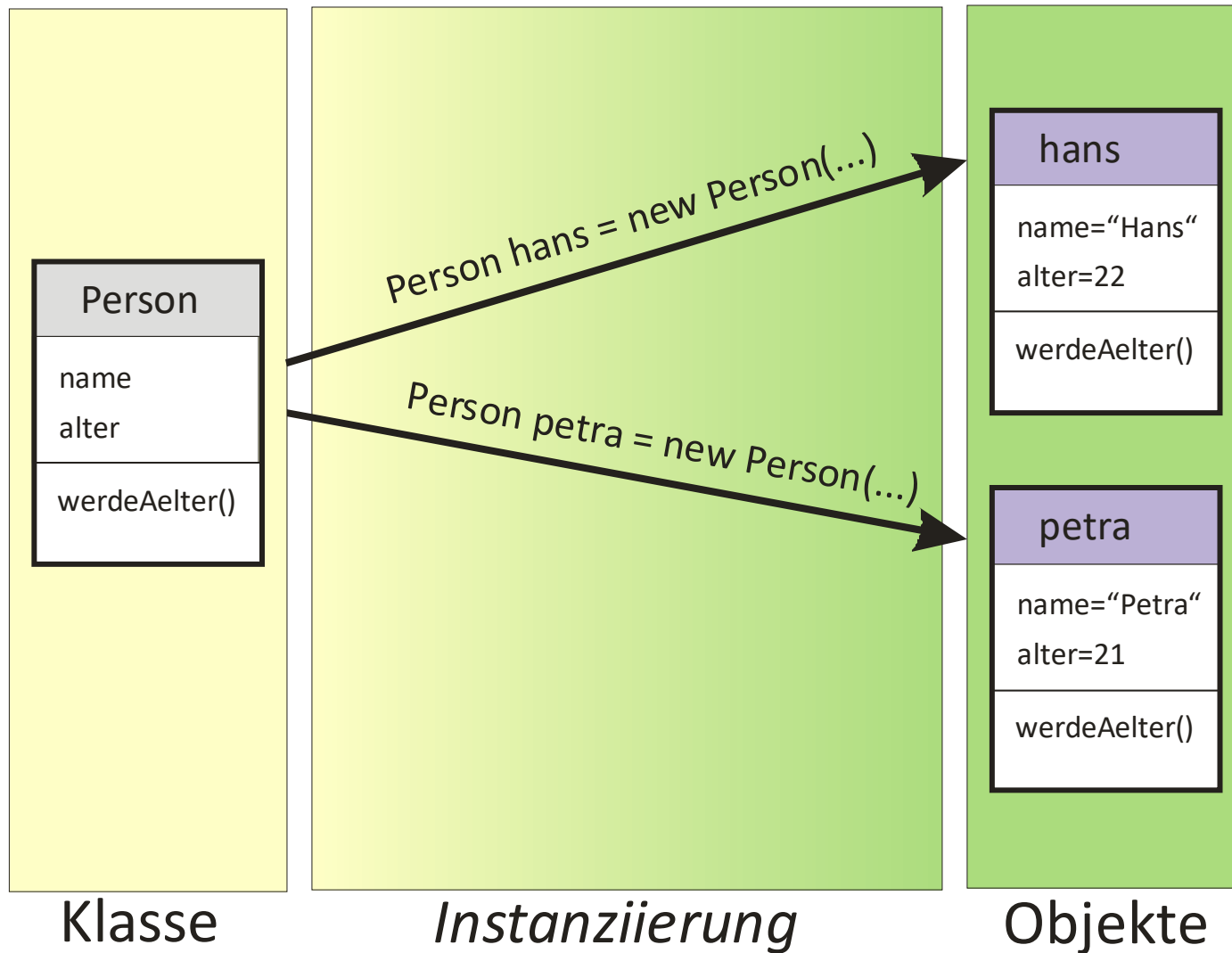
### ▶ Erzeugen:

```
Girokonto einKonto= new Girokonto();
```

### ▶ Löschen:

- ▶ in Java: wenn Objekt nirgendwo mehr benutzt wird  
→ **automatische** Garbage-Collection

# Alternatives Beispiel



# Beispiel: Klasse Girokonto

```
public class Girokonto {
    private int kontostand;    // in Cent gerechnet

    public Girokonto() {
        kontostand = 0;
    }

    /* Einzahlen und Abheben */
    public void zahle (int cent){
        kontostand = kontostand + cent;
    }

    public int holeKontostand() {
        return (kontostand);
    }
}
```

EINI LogWing /  
WiMa

## Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

In diesem Kapitel:

- Prolog
- Einführung

# Beispiel: Wie können wir die Klasse testen?

- ▶ Testen der Klasse "Girokonto" (siehe: Methode "main" in der Klasse "TestGirokonto"):
  - ▶ Erzeugen eines Girokonto-Objekts
  - ▶ Nacheinander:
    - Einzahlen von 100 Cent,
    - Abheben von 20 Cent und
    - Einzahlen von 30 Cent
  - ▶ Anschließend Kontostand ausgeben



# Beispiel – Testklasse

```
01 public class TestGirokonto {
02
03     public static void main(String[] args) {
04
05         Girokonto einKonto = new Girokonto(); // erzeuge neues
06                                                // Konto
07
08
09         einKonto.zahle(100); //Transaktion Einzahlung
10         einKonto.zahle(-20); //Transaktion Auszahlung
11         einKonto.zahle(30); //Transaktion Einzahlung
12
13                                     //gib Kontostand aus
14         int aktuellerStand = einKonto.holeKontostand(); //holen
15
16         System.out.print(aktuellerStand); //anzeigen
17
18     }
19 }
```

## Motivation:

- ▶ Klasse um Attribute und Methoden erweitern
- ▶ Erzeugen von Objekten und Nachrichtenaustausch (Methoden) zwischen Objekten

## Aufgabe:

- ▶ Ergänzung der Funktionalität der Klasse "Girokonto"
  - ▶ zum Sperren eines Kontos: boolesches Attribut **istGesperrt** mit den Methoden **void sperre()**, **void entsperre()** und **boolean istGesperrt()**
  - ▶ zum Überschreiben des aktuellen Kontostandes mit einem übergebenen Betrag: die Methode **setzeKontostand()**

# Beispiel: Erweiterungen

```
01 public class Girokonto {
02     ...
03     private boolean istGesperrt;
04     ...
05
06     public void sperre() {
07         istGesperrt = true;
08     }
09
10     public void entsperre(){
11         istGesperrt = false;
12     }
13
14     public boolean istGesperrt(){
15         return istGesperrt;
16     }
17     ...
18 }
```

EINI LogWing /  
WiMa

## Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

In diesem Kapitel:

- Prolog
- Einführung

## Zwischenstand

- ▶ **Programm** = Mehrere Objekte, die Nachrichten austauschen
- ▶ **Klassen:** Schablonen für Objekte
  - ▶ Attribute
  - ▶ Methoden
- ▶ **Objekte**
  - ▶ Erzeugen
  - ▶ (Zerstören)
- ▶ **Nachrichtenaustausch**
  - ▶ Aufruf von Methoden eines Objektes (Punkt-Notation), z.B. `einKonto.holeKontostand()`

## Unterschiede zu imperativer Programmierung

- ▶ Objekte:
  - ▶ Attribute & Methoden zusammen
  - ▶ vorher in Structs/Records mit Zeigern auf Funktionen auch möglich, aber unüblich
  
- ▶ Klassen:
  - ▶ vorher als Module, in denen Funktionen für bestimmte Datenstrukturen gesammelt wurden
  - ▶ auch in imperativer Programmierung möglich
  
- ❖ Der wesentliche Unterschied ist eine andere, nämlich objektorientierte Herangehensweise, die durch Sprachkonstrukte unterstützt wird.

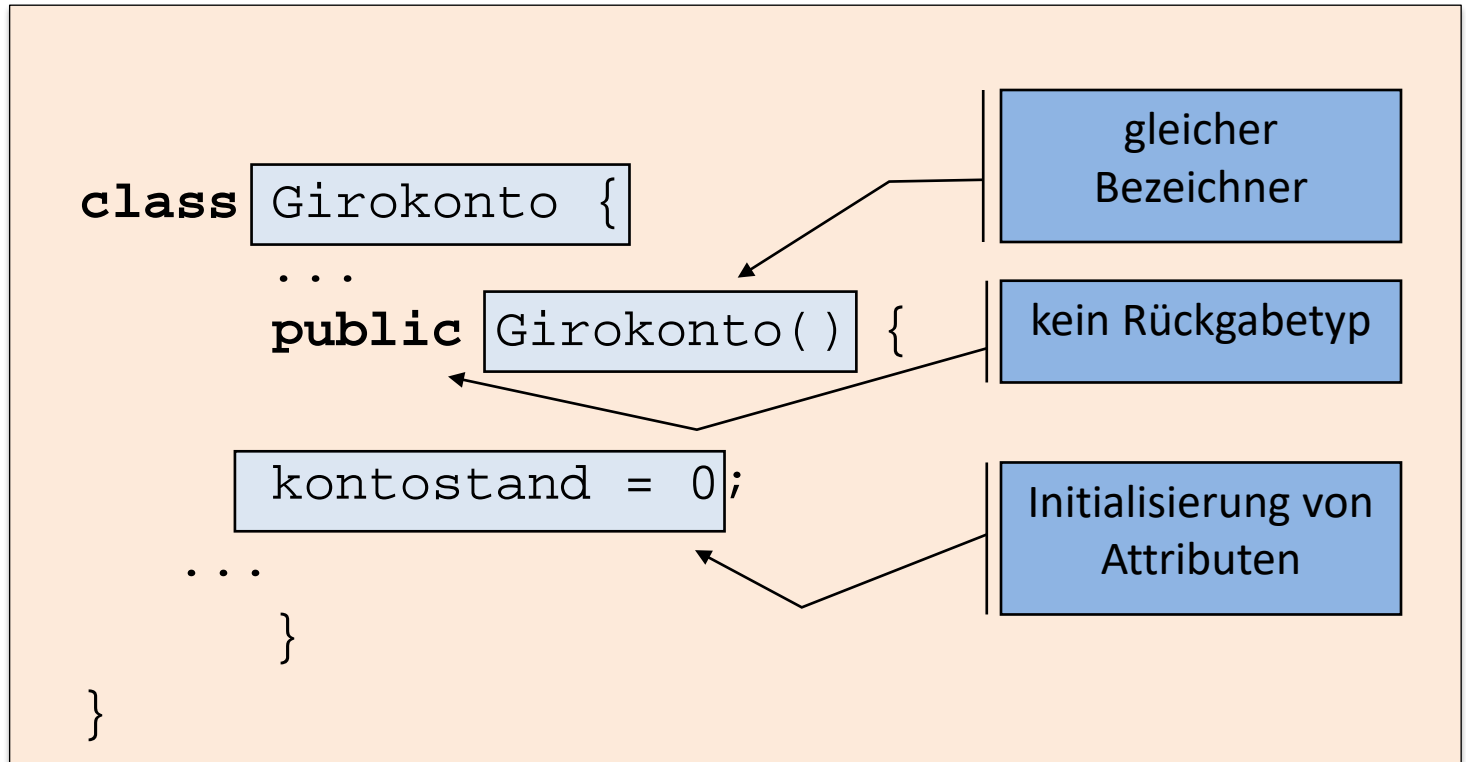
## Übersicht

- ▶ Konstruktoren: Aktivitäten bei Objekterzeugung
- ▶ Referenzen als Verweise auf Objekte
- ▶ Übergabe von Parametern an Methoden
- ▶ Klassenattribute / Klassenmethoden
- ▶ Namensraum
  - ▶ Überladen von Methoden
  - ▶ Überdecken von Attributen

# Details: Konstruktor

- ▶ **Methode**, die automatisch bei Erzeugung eines Objektes aufgerufen wird.
- ▶ Wird in der Regel benutzt, um Attribute zu initialisieren.
- ▶ Dadurch charakterisiert, dass der **Methodenname** mit dem **Klassennamen** übereinstimmt.
- ❖ Konstruktoren besitzen **keinen Rückgabewert**, auch nicht `void`!

# Details: Konstruktor (Beispiel)



EINI LogWing /  
WiMa

## Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

### In diesem Kapitel:

- Prolog
- Einführung



# Details: Konstruktor

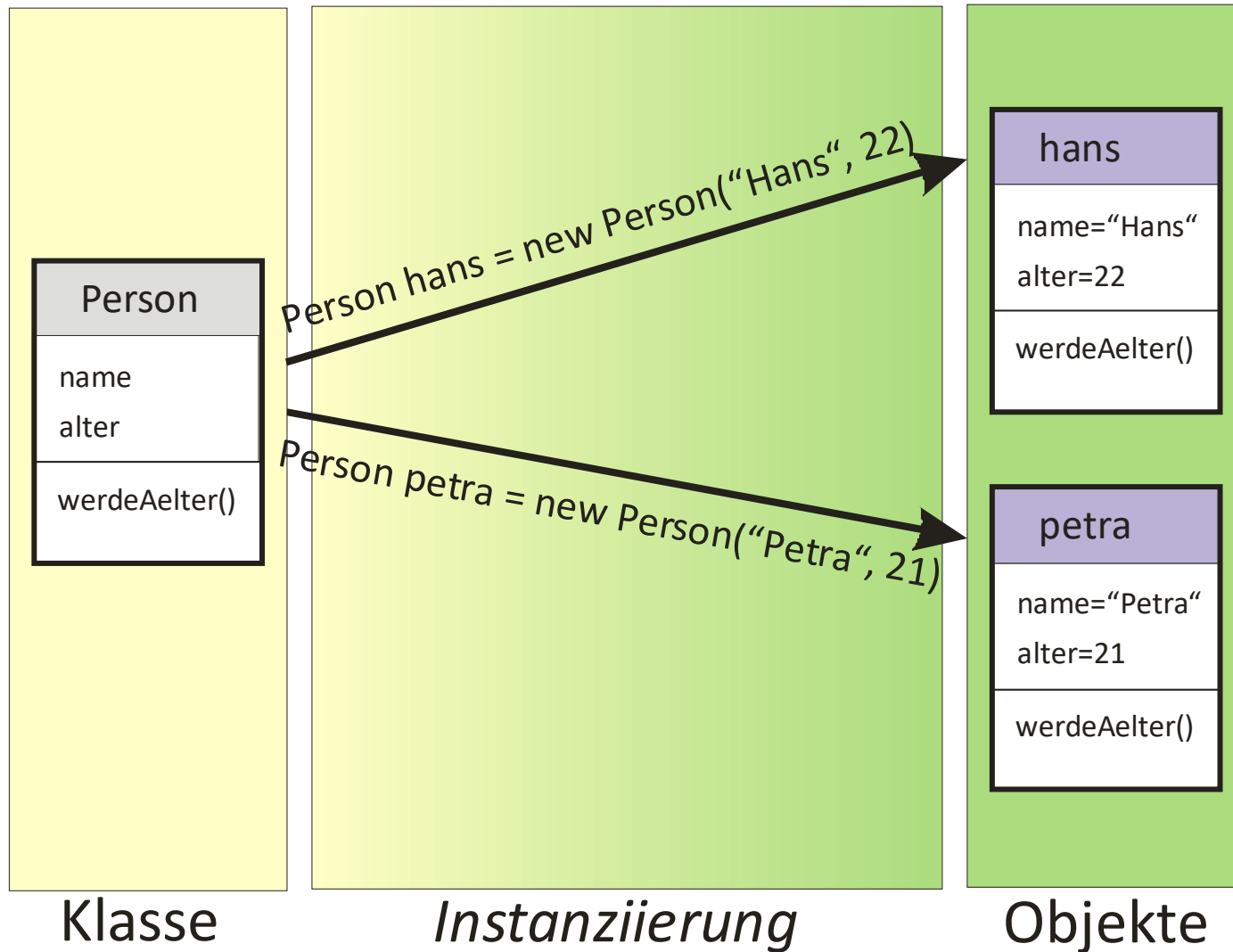
- ▶ Argumente der Konstruktormethode werden bei Objekterzeugung durch

```
new Klassenname (Argument1 , Argument2 , . . . )
```

an den Konstruktor übergeben.

- ▶ Anweisungen im Konstruktor werden auf neu zugewiesenem Speicher ausgeführt.
- ▶ Ohne Angabe eines Konstruktors wird automatisch ein leerer Konstruktor ohne Argumente definiert.
- ▶ Ist ein (nicht leerer) Konstruktor definiert, muss der leere Konstruktor explizit definiert werden, um noch benutzt werden zu dürfen.

# Alternatives Beispiel



# Details: Referenzen auf Objekte I

- ▶ Bei primitiven Datentypen enthält eine Variable direkt den Inhalt (z. B. einen Int-Zahlenwert).
- ▶ Bei Objekten von Klassen symbolisiert die Variable nur eine **Referenz** (einen Verweis) auf das Objekt.
- ▶ Es können auch mehrere Referenzen auf ein Objekt zeigen.
- ▶ Eine Referenz kann auch leer sein. **null** ist das Schlüsselwort für die leere Referenz.
  - ▶ Beispiel: `Girokonto k = null;`

# Details: Referenzen auf Objekte II

Mit dem `==` Operator kann man zwei Referenzen auf **Gleichheit** testen.

## ▶ Beispiel

```
Girokonto k1 = new Girokonto();  
  
...  
  
Girokonto k2 = k1;  
  
...  
  
if (k1 == k2)  
    System.out.println("k1,k2 verweisen auf  
                        das selbe Objekt");
```

# Details: Referenzen auf Objekte III

Mit dem `==` Operator kann man auch abfragen, ob eine Referenz leer ist.

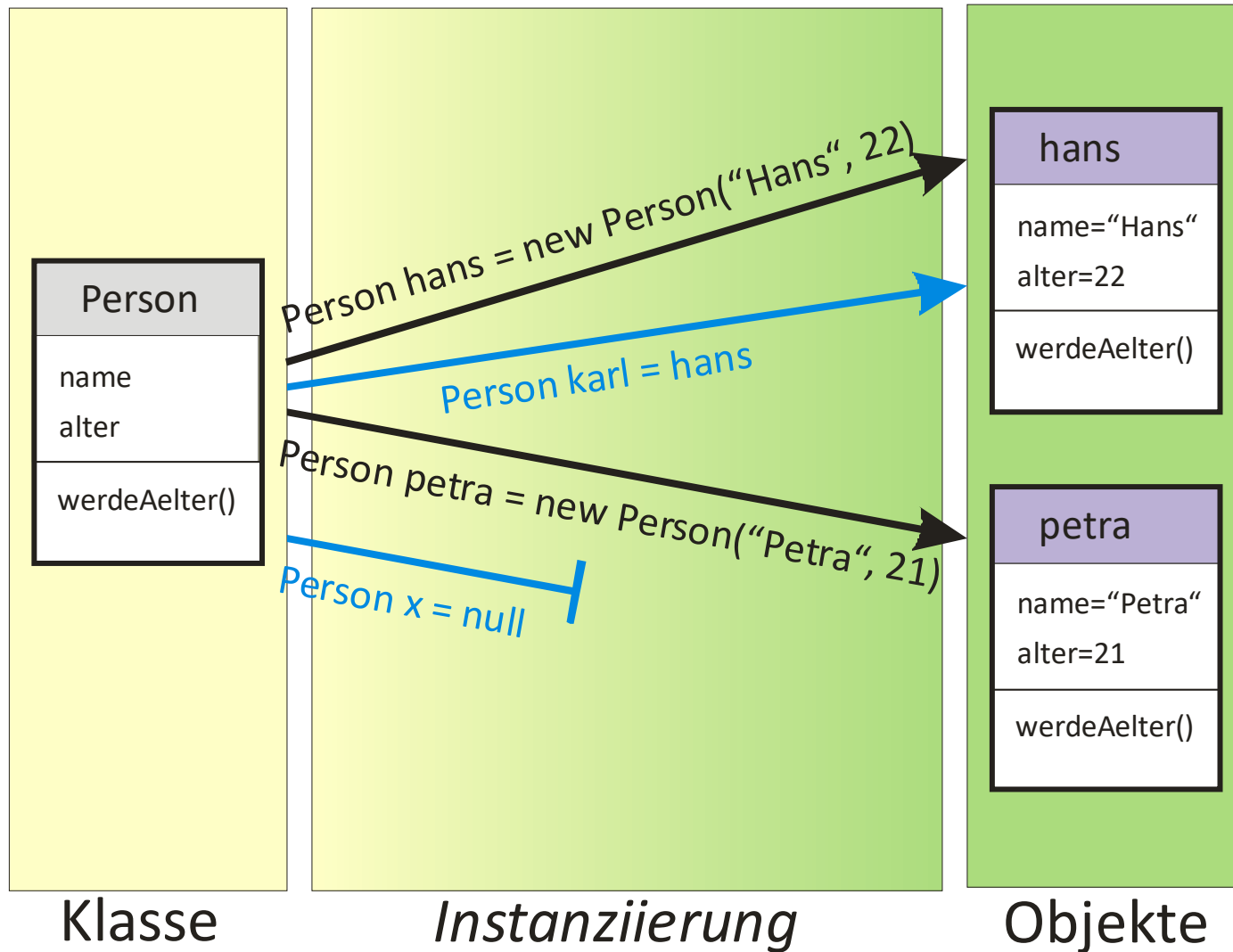
## ▶ Beispiel

```
Girokonto k=null;

...

if (k == null)
    System.out.println("k ist leere Referenz");
```

# Alternatives Beispiel



# Details: Klassenattribute und -methoden

- ▶ **Attribute** und **Methoden** gehören zu den Objekten (also Instanzen) einer Klasse.
- ▶ Attribute, die für jedes Objekt neue Instanzen bilden, heißen daher **Instanzvariablen** (Standardfall).
- ▶ Aber:
  - ▶ Es gibt **Klassenattribute** und **-methoden**, die nicht zu Instanzen (also Objekten) gehören,
  - ▶ sondern zu den Klassen selbst.

# Details: Klassenvariablen

- ▶ Attribute, die für jedes Objekt neue Instanzen bilden, heißen **Instanzvariablen** (Standardfall)
- ▶ Werden Attribute mit **static** gekennzeichnet, handelt es sich um **Klassenvariablen**, die für die gesamte Klasse nur eine Instanz besitzen
- ▶ Klassenvariablen existieren auch ohne die Existenz eines Objektes.
  - ▶ Zugriff durch *Klassenname.Attributname*



# Details: Klassenmethoden

- ▶ Methoden,
  - ▶ die ausschließlich auf Klassenvariablen zurückgreifen,
  - ▶ dürfen mit **static** gekennzeichnet werden.
- ▶ Diese heißen **Klassenmethoden**.
  - ▶ Klassenmethoden dürfen selbst auch nur Klassenmethoden benutzen.
- ▶ Klassenmethoden können auch ohne Existenz eines Objektes mit *Klassenname.Methodenname(...)* aufgerufen werden.
- ❖ Die **main( )**-Methode ist eine Klassenmethode, da zu Beginn noch keine Objekte erzeugt wurden.

# Details: Beispiel

```
class Demo {  
  
    int a;  
    static int b;  
  
    void test() {  
        a = 2;  
        b = 2;  
    }  
  
    static void test2() {  
        b = 3;  
    }  
}
```

Instanzvariablen  
Instanzmethoden  
Klassenvariablen  
Klassenmethoden

```
public static void main (String[] args) {  
    a = 1;        // nicht erlaubt  
    b = 1;        // erlaubt  
    test();       // nicht erlaubt  
    test2();      // erlaubt  
  
    Demo d = new Demo();  
  
    d.a = 1;      // erlaubt  
    d.b = 1;      // erlaubt  
  
    Demo.a = 1;   // nicht erlaubt  
    Demo.b = 1;   // erlaubt  
  
    Demo.test();  // nicht erlaubt  
    Demo.test2(); // erlaubt  
  
}
```

# Details: Überladen von Methoden I

- ▶ Wünsche für Methoden/Attribute:
  - ▶ gleiche Namen für gleiche Zwecke
  - ▶ z.B. Girokonto erzeugen ohne/mit einem initialen Kontostand
- ▶ Benutzt werden sollen Methoden mit **gleichem Namen**.
- ▶ Problem: Wie soll man diese Methoden **auseinanderhalten**?
- ▶ Idee: Unterscheide anhand **Parameter-Typen**:
  - ▶ Methoden gleichen Namens müssen sich also im Typ von mindestens einem Parameter oder in der Anzahl der Parameter unterscheiden:

```
zahle (int betrag)
```

```
zahle (int betrag, String verwendungszweck)
```

# Details: Überladen von Methoden II

- ❖ Unterschiedliche Methoden müssen sich im Methodennamen oder in der Übergabeparameterliste (oder beidem) unterscheiden.
- ▶ Hat eine Klasse mehrere Methoden mit **identischem** Namen, nennt man diese Methode **überladen**.
- ▶ In unterschiedlichen Klassen dürfen auch
  - ▶ Methoden mit identischem Namen und identischen Übergabeparameterlisten
  - ▶ deklariert werden.
- verschiedene Namensräume!

EINI LogWing /  
WiMa

## Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

### In diesem Kapitel:

- Prolog
- **Einführung**

# Details: Überdecken von Attributen I

Verwenden von Variablen mit bereits benutztem Namen:

- ▶ Zugreifen auf überdeckte Attribute über **this**:

```
public class Girokonto {
    private int kontostand;    // in Cent

    /*...*/

    public void setzeKontostand(int kontostand)
    {
        this.kontostand = kontostand;
    }
    /*...*/
}
```

# Details: Überdecken von Attributen II

- ▶ Variablendeklarationen in Klassenmethoden überdecken die Attribute der Klasse.
- ▶ Die Attribute sind nur **überdeckt, nicht überschrieben**.
- ▶ Auf Attribute der Klasse kann dann über das Schlüsselwort **this** zugegriffen werden.
- ▶ **this** ist eine Referenz auf das zur Methode gehörende Objekt.

EINI LogWing /  
WiMa

## Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

### In diesem Kapitel:

- Prolog
- **Einführung**

# Details: Beispiel I

## Überladen von Konstruktoren und Methoden:

- ▶ Erweitern einer Klasse Girokonto um Attribute und Methoden.
- ▶ Ziel: **Übersicht behalten** durch sinnvolles, sparsames Vergeben von Methodennamen (Namen mehrfach in einer Klasse vergeben).

## Anforderung:

- ▶ Implementieren einer zweiten Konstruktormethode für die Klasse „Girokonto“.
  - ▶ Sie soll es ermöglichen, dass schon bei der Erzeugung eines Girokonto-Exemplars ein Wert für den initialen Kontostand übergeben werden kann.
  - ▶ Welchen Namen muss diese Methode tragen?
  - ▶ Der übergebene Parameter soll **kontostand** heißen.
  - ▶ Es gibt in der Klasse „Girokonto“ bereits ein Attribut mit diesem Namen.

# Details: Beispiel II

## Überladen einer Konstruktormethode

▶ Lösungsmuster:

- ▶ Konstruktormethode → Name identisch mit Klassenname
- ▶ Überdecken des Namens `kontostand` (als Attributname und Variablenname)

```
public class Girokonto {  
  
    /* ... */  
  
    public Girokonto(int kontostand) {  
        this.kontostand = kontostand;  
    }  
  
    /* ... */  
  
}
```



# Zwischenstand I

- ▶ Programm = Mehrere Objekte, die Nachrichten austauschen
- ▶ Klassen: Schablonen für Objekte
  - ▶ Attribute
  - ▶ Methoden
- ▶ Objekte
  - ▶ Erzeugen
  - ▶ Zerstören
- ▶ Nachrichtenaustausch
  - ▶ Aufruf von Methoden eines Objektes (Punkt-Notation)  
z.B. `einKonto.holeKontostand()`

EINI LogWing /  
WiMa

## Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

### In diesem Kapitel:

- Prolog
- **Einführung**

- ▶ Einzelheiten zu:
  - ▶ Konstruktoren
  - ▶ Referenzen
  - ▶ Garbage Collection
  - ▶ Übergabe von Parametern an Methoden
  - ▶ Klassenattributen / Klassenmethoden
  
- ▶ Namensraum
  - ▶ Überladen von Methoden
  - ▶ Überdecken von Attributen



# Objektorientierte Programmierung – Einführung

Artikel im EINI-Wiki:

- **Objektorientierte Programmierung**
- **Klasse**
- **Modifikator**
- **Sichtbarkeit**
- **Main**
- **Objekt**
- **Garbage Collector**
- **Konstruktor**
- **Referenz**
- **Null**
- **Variable** (→ Klassenvariablen)
- **This**

## Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

In diesem Kapitel:

- Prolog
- **Einführung**

# Übersicht

## Begriffe

- ✓ Spezifikationen, Algorithmen, formale Sprachen
- ✓ Programmiersprachenkonzepte
- ✓ Grundlagen der imperativen Programmierung
  
- ✓ Algorithmen und Datenstrukturen
  - ✓ Felder
  - ✓ Sortieren
  - ✓ Rekursive Datenstrukturen (Baum, binärer Baum, Heap)
  - ✓ Heapsort
  
- Objektorientierung
  - ✓ Einführung
  - Vererbung
  - ▶ Anwendung

EINI LogWing /  
WiMa

### Kapitel 6

Objektorientierte  
Programmierung -  
Einführung

In diesem Kapitel:

- Prolog
- Einführung

# Übersicht



© Eucalyp, <https://creativemarket.com/eucalyp>

## Vielen Dank für Ihre Aufmerksamkeit!

### Nächste Termine (über ZOOM)

- ▶ Große Übung – WiMa 22.12.2022, 08:15
- ▶ Große Übung – LogWing 23.12.2022, 08:15