Software-Entwicklung 1

V10: Arrays und Klassenmethoden

Prof. Maalej & Team **@maalejw**

Status der 9. Übungswoche

Zeit	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
Vormittag	Gruppe 1 **Erfüllt: 70%**	Gruppe 3 **Erfüllt: 62%**	Gruppe 5 **Erfüllt: 68%**	Gruppe 6 **Erfüllt: 63%**	Gruppe 8 **Erfüllt: 63%**
Nachmittag	Gruppe 2 **Erfüllt: 79%**	Gruppe 4 **Erfüllt: 62%**	Vorlesung	Gruppe 7 **Erfüllt: 47%**	

Tutorium Level 3

- Michael Strassberger

- Heute
- 21.12.16

- 18:30 Uhr

- D-018

Überblick

1. **Arrays**
2. Klassenmethoden

Beispiel: Temperaturmessung

- Temperatursensor in einer Boje
- Sensoren messen den ganzen Tag über immer wieder die Temperatur
- Der Speicher ist auf 1000 Messwerte begrenzt
- Wir wollen
  - die letzten 1000 Messungen speichern,
  - Maximum und Minimum finden

Lösung?

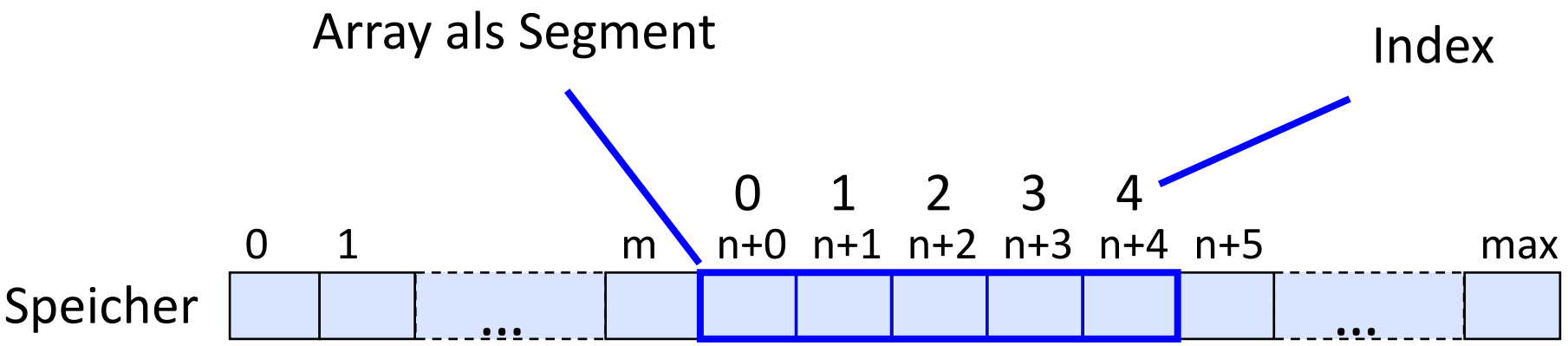
```
class Boje
{
	private int _messung1;
	private int _messung2;
	...
	private int _messung100;

	public int gibMaximum()
	{
		...
	}
}
```

Einordnung von Arrays

- Referenztypen
  - Java Typen
    - String
    - **Arrays**
  - Eigene Typen
    - Konto
    - ...
- Primitive Typen

Arrays sind ein speichernahes Konzept

- **Sammlung gleichartiger** Elemente
- Zugriff erfolgt über einen **Index**
- Listen mit **fester Größe**
- Sie abstrahieren von einem **zusammenhängenden Speicherbereich** mit **indiziertem** Zugriff auf die Speicherzellen



Arrays für Bilddaten

- Bilder sind digital zweidimensionale Strukturen einzelner **Bildpunkte**
- Bildgröße (Zeilen und Spalten) ist statisch
- Informationen über Bildpunkte müssen für Bildverarbeitung **schnell** zugreifbar sein

Arrays für Spielfelder

Tic-Tac-Toe: 3x3

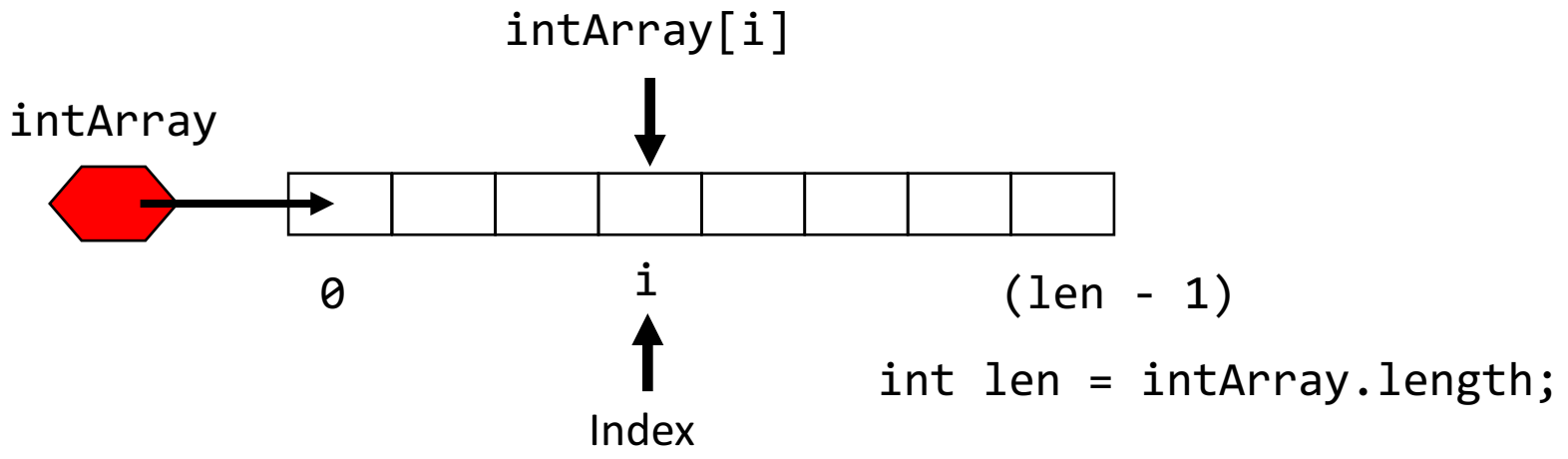
Vier Gewinnt: 6x7

Schach: 8x8

Übersicht: Arrays in Java

- **Arrays** in Java:
  - Eine **geordnete Reihung gleichartiger Elemente**
  - Elementtypen können **Basistypen** oder **Referenztypen** sein (auch Referenzen auf andere Arrays)
  - Die **Länge eines Array** wird erst beim Erzeugen festgelegt
  - Jeder **Zugriff** über den Index wird **automatisch geprüft**

```
int [] intArray = new int[8];
```

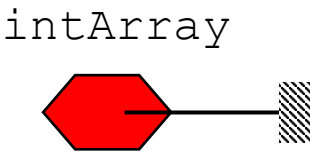


Arrays Objekte

- Ein Array ist in Java immer ein **Objekt** (Arrays haben alle Eigenschaften, die in der Klasse **Object** definiert sind)
- Eine **Array-Variable** ist immer eine **Referenzvariable** (Typ ist „Array von Elementtyp“)
- Beispiel einer **Deklaration** eines Arrays von Integer-Werten:

```
int[] intArray;
```

- Die **Länge/Größe** des Arrays wird in der Deklaration **nicht** angegeben



Syntax: Array-Deklaration


```
Type [] Identifier
```

Beispiele:

```
// Array-Variable mit primitivem Elementtyp („Array von int“)
int[] numbers;

// Array-Variable mit einem Objekttyp als Elementtyp („Array von
   Person“)
Person[] people;
```

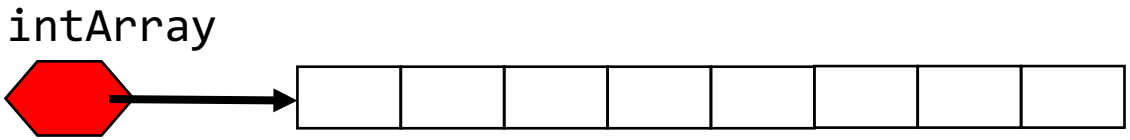
Array-Erzeugung

- **Array-Objekte** müssen explizit **erzeugt** werden (wie alle Objekte mit **new**)
- Länge wird definiert definiert oder berechnet und bleibt unverändert

```
intArray = new int [8];
```

- **Deklaration** und **Erzeugung** können **zusammengefasst** werden:

```
int[] intArray = new int [8];
```



Syntax: Array-Erzeugung

- Arrays werden mit dem Schlüsselwort **new** erzeugt:

```
new Type [ LengthExpression ]
```

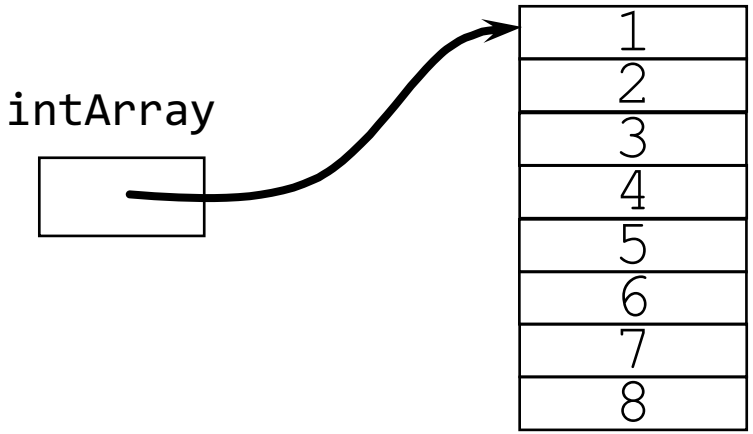
- Beispiele für Ausdrücke:

```
// Erzeugung eines Arrays mit primitivem Elementtyp
new int[10]

// Erzeugung eines Arrays mit einem Objekttyp als Elementtyp
new Person[x]
```

Initialisieren von Array-Zellen

- Bei Array-Erzeugungen erhalten die **Zellen** eines Arrays in Java die Default-Werte des Elementtyps
- Neben der normalen Zuweisung von Werten kann ein Array auch implizit **erzeugt** und **direkt initialisiert** werden

```
int[] intArray = {1,2,3,4,5,6,7,8};
```



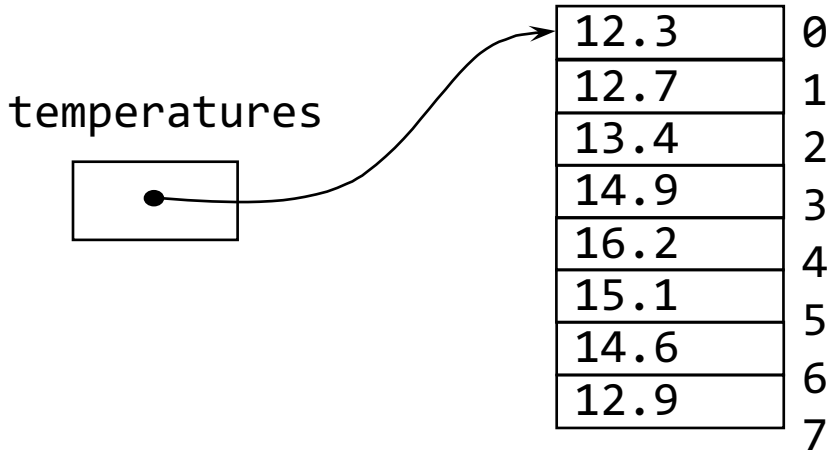
Diese implizite Erzeugung und Initialisierung mit **geschweiften Klammern** ist ausschließlich bei der Deklaration erlaubt!

Indizierung bei Arrays

- Arrays werden **beginnend bei 0** indiziert
- Gültige Indizes eines Arrays sind **ganzzahlig** von 0 bis length – 1
- Beispiel: Ein Array der Größe 8 hat als gültige Indizes 0..7

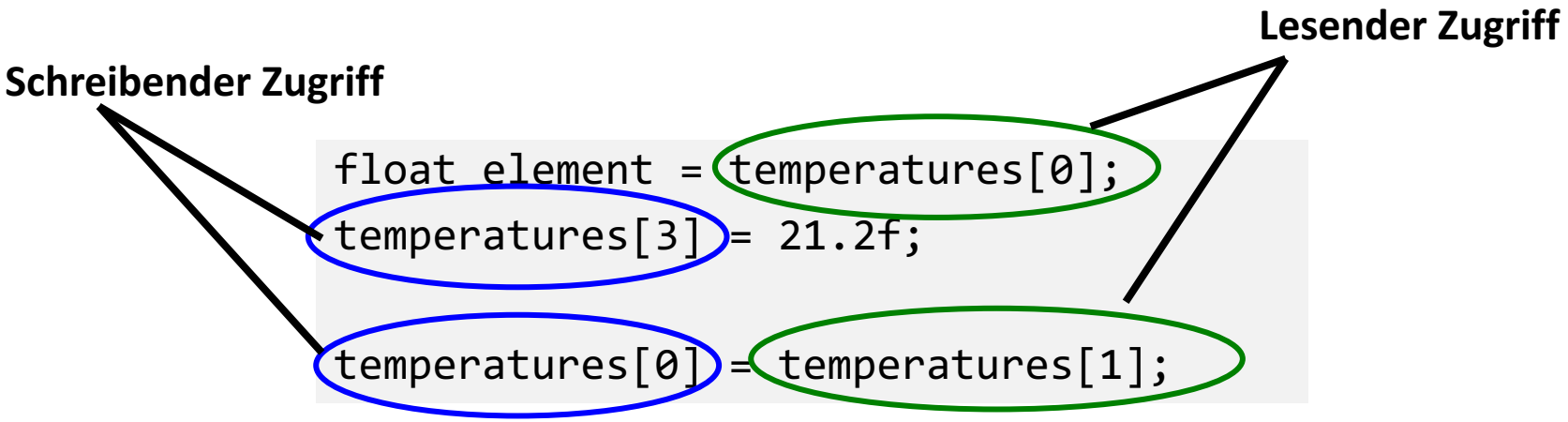
Gültige Namen für die Elemente des Arrays sind:

- `temperatures[0]`
- `temperatures[1]`
- `...`
- `temperatures[7]`

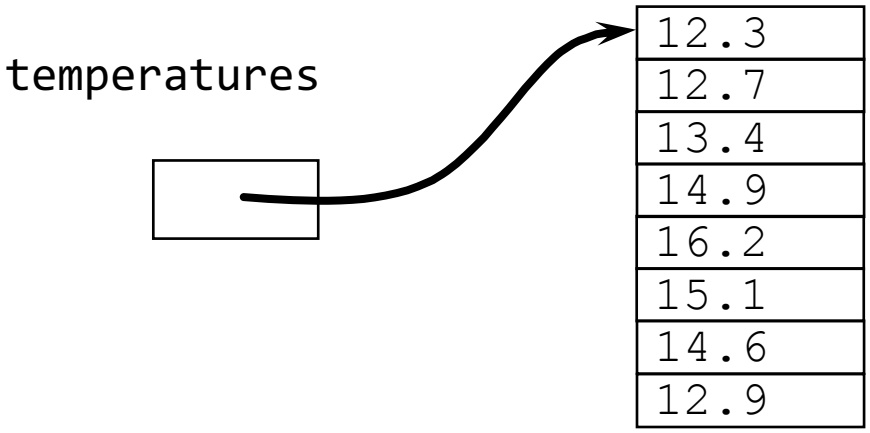


Schreibender und lesender Zugriff auf Array-Zellen

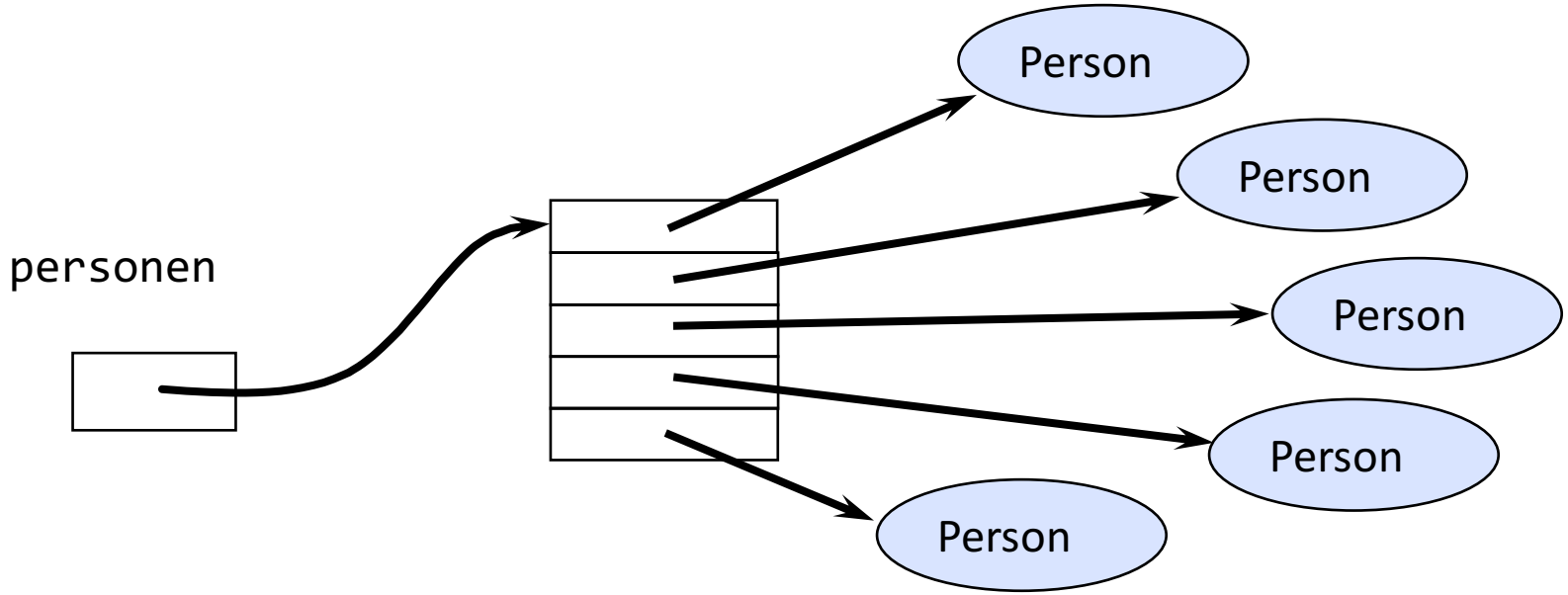
- Auf die Array-Zellen wird mit eckigen Klammern **[]** zugegriffen:



Werte und Objekte als Elemente

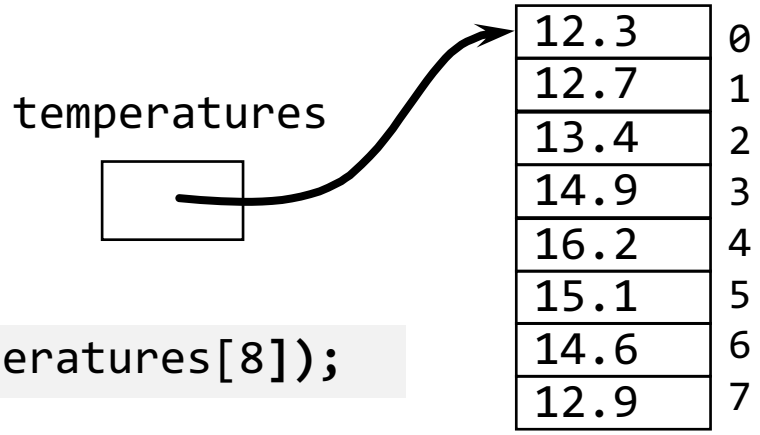


- Array-Zellen speichern **Werte** oder **Referenzen auf Objekte** genau wie andere Variablen auch



Typischer Fehler

- Angenommen, wir haben ein Array der Größe 8
- Dann schreiben wir:




```
System.out.println(temperatures[8]);
```

- Es kommt zu einer Fehlermeldung:

`ArrayIndexOutOfBoundsException: 8`

Typischer Fehler

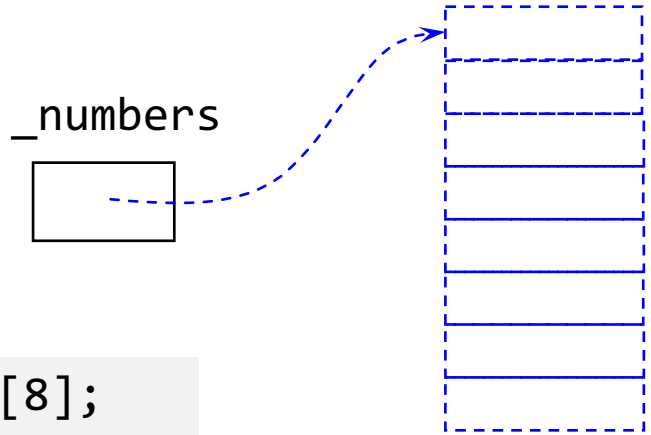
- Angenommen, wir haben ein Array als Exemplarvariable deklariert:

```
private int[] _numbers;
```

- Im Konstruktor schreiben wir als Erstes:

```
_numbers[0] = 42;
```

- Es kommt zu einer Fehlermeldung: `NullPointerException`



- Es fehlt die **Erzeugung** des Arrays:

```
_numbers = new int[8];
```

Typischer Fehler

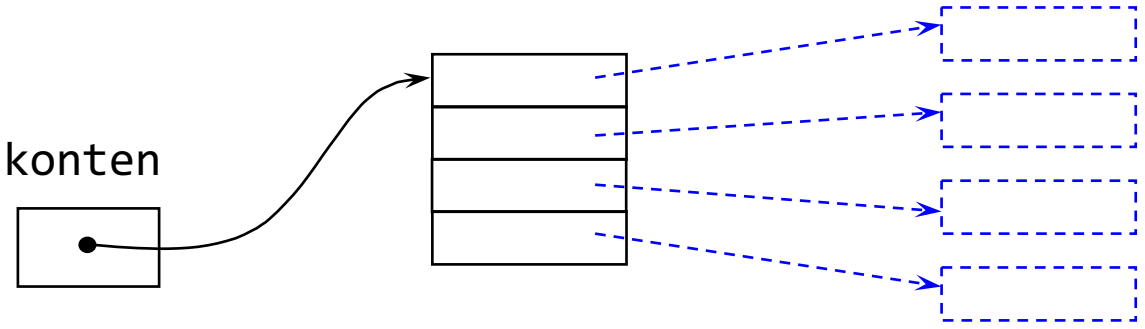
- Angenommen, wir haben ein Array mit einem Objekttyp als Elementtyp deklariert und initialisiert:

```
Konto[] konten = new Konto[4];
```

- Unmittelbar danach schreiben wir:

```
konten[0].einzahlen(123);
```

- Es kommt zu einer Fehlermeldung: `NullPointerException`



- Es fehlt die **Erzeugung der Elemente** des Arrays, also der Konto-Objekte

For-Schleifen und Arrays

- Typischerweise werden For-Schleifen eingesetzt, um alle Elemente eines Arrays zu bearbeiten
- Dabei wird die **öffentliche Exemplarkonstante length** benutzt

- Beispiel: Das Ausgeben der Werte eines Arrays

```
public void printArray(int[] intArray)
{
    for (int i = 0; i < intArray.length; ++i)
    {
        System.out.println(intArray[i]);
    }
}
```

Eine solche Standardbenutzung einer For-Schleife für Arrays wird auch als **Programmiermuster** bezeichnet. Im Englischen wird oft der Begriff **idiom** verwendet.

Erweiterte For-Schleife für Arrays

- Die erweiterte For-Schleife kann für Arrays verwendet werden
- Dies erspart den Zugriff auf **length**
- Beispiel: Das Ausgeben der Werte eines Arrays

```
public void printArray(int[] intArray)
{
    for (int k : intArray)
    {
        System.out.println(k);
    }
}
```

Lies: für jeden int k im intArray tue…

Diese Schleifenart ist nicht geeignet, wenn an der **Belegung** der Zellen etwas geändert werden soll. Es steht im Schleifenrumpf **kein Schleifenindex** zur Verfügung.

Beispiel: Den minimalen Wert finden

```
/**
 * Liefere den minimalen Wert im gegebenen Array.
 */
public int findeMinimum(int[] intArray)
{
    int min = Integer.MAX_VALUE;
    for (int i=0; i < intArray.length; ++i)
    {
        if (intArray[i] < min)
        {
            min = intArray[i];
        }
    }
    return min;
}
```

Alternativ: neue For-Schleife

```
{
    int min = Integer.MAX_VALUE;
    for (int k : intArray)
    {
        if (k < min)
        {
            min = k;
        }
    }
    return min;
}
```

Benutzung:

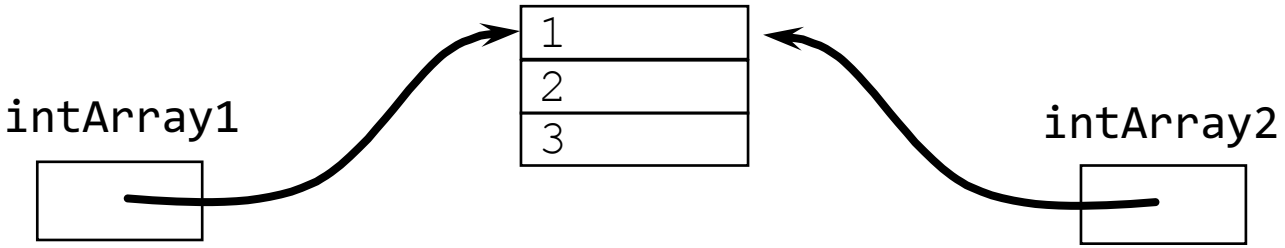
```
int[] myArray = new int[10];
myArray[0] = 20;
myArray[1] = 40;
myArray[2] = 30;
int mini = findeMinimum(myArray);
System.out.println(mini);
```

Zuweisungen mit Arrays

- Die Zuweisung einer Array-Variablen kopiert **nur eine Referenz**!
- Beispiel:

```
int[] intArray1 = { 1, 2, 3 };
int[] intArray2 = intArray1;
```

- Beide Referenzen verweisen nun auf **dasselbe Array-Objekt**:

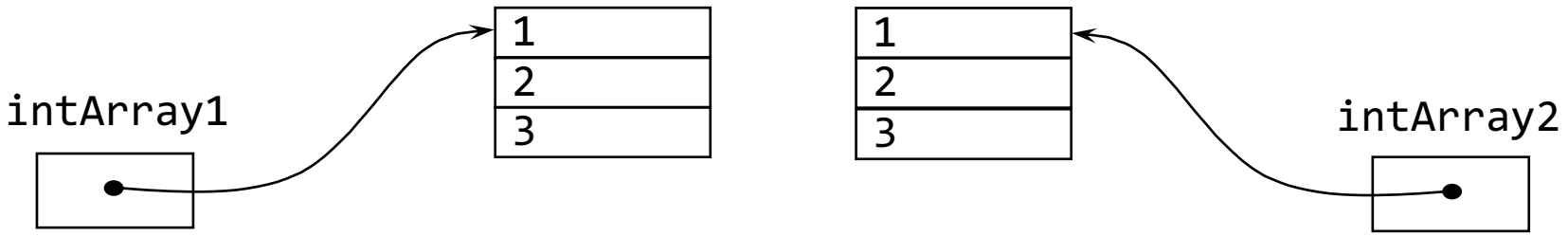


- Für eine **Kopie des Array-Objektes** gibt es zwei Möglichkeiten:
  - Elementweise in ein neues Array-Objekt kopieren
  - Die Operation **clone** verwenden

Kopieren von Array-Objekten

- Neues Array-Objekt selbst erzeugen und elementweise kopieren:

```
int[] intArray1 = { 1, 2, 3 };
int[] intArray2 = new int[intArray1.length];
for (int i=0; i < intArray1.length; ++i)
{
    intArray2[i] = intArray1[i];
}
```

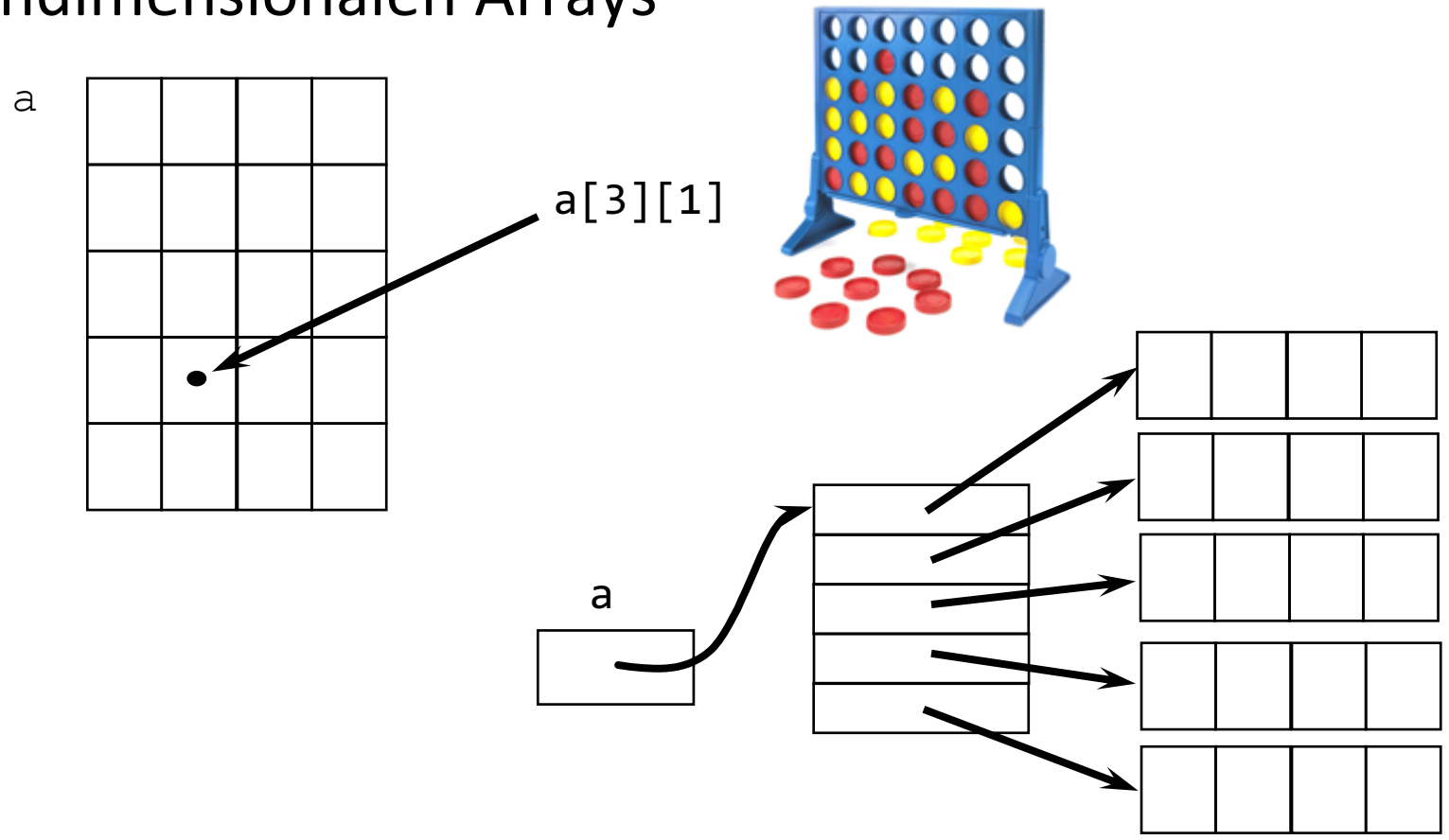


- Die Operation **`clone`** verwenden:

```
int[] intArray1 = { 1, 2, 3 };
int[] intArray2 = intArray1.clone();
```

Zweidimensionale Arrays

- Zweidimensionale Arrays in Java sind Arrays von eindimensionalen Arrays

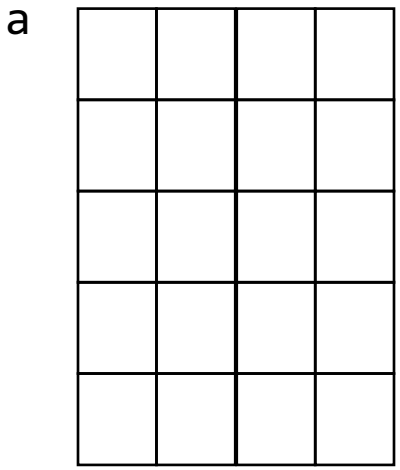


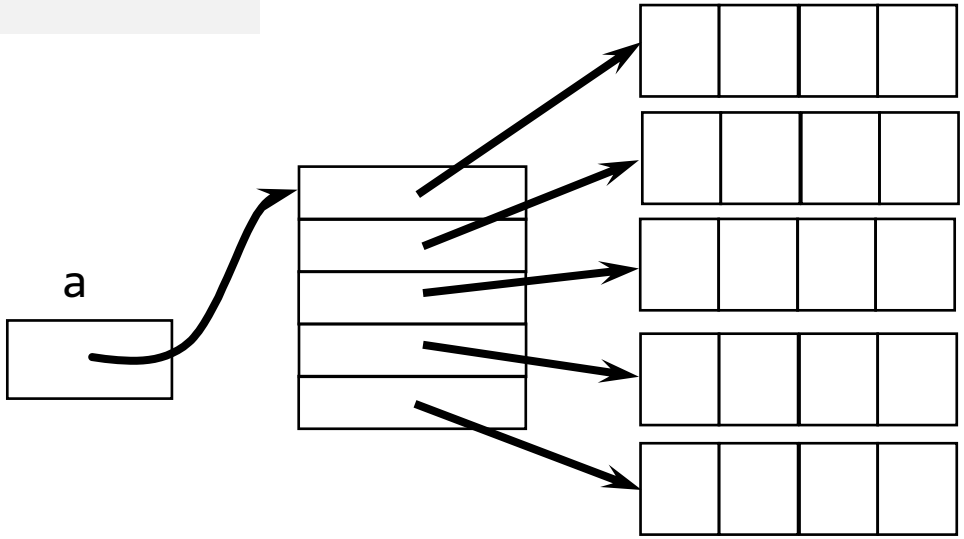
- a eine Referenz auf ein Array von Zeilen; der erste Index benennt die Zeile, der zweite die Spalte

Zweidimensionale Arrays erzeugen

```
int[][] a;
a = new int[5][];
for (int i=0; i<5; ++i)
{
   a[i] = new int[4];
}
```

Mit new ist es auch möglich, ein zweidimensionales Arrays direkt zu erzeugen.
Also: `new int[5][4]`



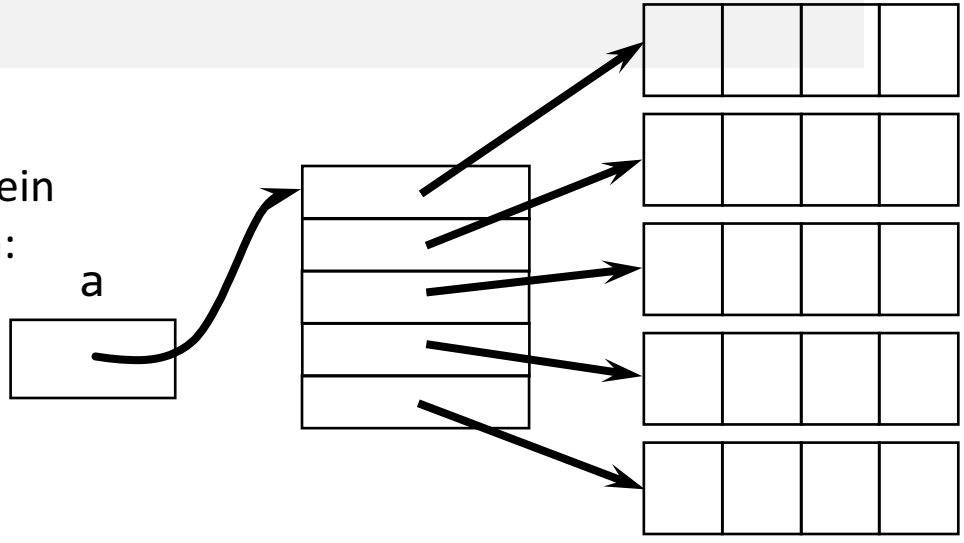


Zugriff auf zweidimensionale Arrays

```
// Alle Array-Elemente auf den Wert 7 setzen

for (int row=0; row < a.length; ++row)
{
   for (int column=0; column < a[row].length; ++column)
   {
        a[row][column] = 7;
   }
}
```

Mit Hilfe von Initializern kann auch direkt ein zweidimensionales Array angelegt werden:

```
int[][] a = { {7,7,7,7},
              {7,7,7,7},
              {7,7,7,7},
              {7,7,7,7},
              {7,7,7,7} };
```



Kopieren von mehrdimensionalen Arrays

- Bei mehrdimensionalen Arrays ist es prinzipiell wie bei eindimensionalen:
  - Manuell kopieren: Für jede Dimension die notwendigen Arrays erzeugen und deren Inhalte kopieren
  - Oder es wird die Operation clone benutzt. Dabei ist zu beachten:
    - **clone** ist für Arrays **nicht rekursiv** implementiert
    - D.h., bei einem Aufruf von clone auf einer Array-Variablen für ein mehrdimensionales Array wird nur das Array auf der obersten Ebene kopiert
    - Wenn eine vollständige Kopie gewünscht ist, dann muss „von Hand“ für alle Dimensionen geklont werden

Vorteile von Arrays

- Effizienzvorteile durch speichernahe Implementation:
  - Elementzugriff kann direkt auf einen **Index-Zugriff** abgebildet werden; dies ist sehr effizient
  - Effiziente **Kopiervorgänge** von Arrays

- **Arrays in Java** haben gegenüber den dynamischen Sammlungen außerdem den Vorteil, dass sie **auch elementare Typen** als Elementtyp zulassen

Nachteile von Arrays

- Ein Array, einmal erzeugt, hat eine **feste Maximalkapazität**

- Auf einem Array gibt es außer dem indizierten Zugriff **keine höherwertigen Operationen** (z.B. einfügen, entfernen, anfügen, testen auf Enthaltensein)

Zusammenfassung

1. Ein **Array** ist eine elementare imperative Datenstruktur, die sehr speichernah konzipiert ist.

2. Array sind geordnete **Reihungen gleichartiger Elemente**, auf die über einen **Index zugegriffen** wird.

3. Für Java gilt: Die **Elemente** können von **elementarem** Typ oder **Referenztypen** sein (auch Referenzen und andere Arrays).

4. Die **Größe** eines Arrays wird erst **beim Erzeugen** festgelegt. Jeder Zugriff über den Index wird zur Laufzeit überprüft.

Überblick

1 Arrays

2 **Klassenmethoden**

Klassen und Objekte - revisited

- **Unterschied zwischen Klasse und Objekt** im objektorientierten Modell:
  - **Klassen** sind die Einheiten des **statischen**
  - **Objekte** sind die Einheiten des **laufenden Programms**

- Wenn **Klassen** selbst vollständig **im Laufzeitsystem verfügbar** sind, verschiebt sich diese klare Unterteilung:
  - Über den Zugriff auf eine Klasse kann zur Laufzeit das Verhalten ihrer Objekte verändert werden
  - Klassen werden zu eigenständigen Objekten mit einem eigenen Zustandsraum

Klassen in Java sind auch selbst Objekte

- Klassen existieren auch selbst als Objekte zur Laufzeit
- Ein solches **Klassenobjekt** kann wie alle Objekte einen Zustand haben (über **Klassenvariablen**) und Methoden anbieten (**Klassenmethoden**)
- Klassenvariablen und Klassenmethoden werden mit dem Modifikator **`static`** deklariert

```
class Konto
{
    private static int exemplarzaehler = 0;

    public Konto()
    {
        exemplarzaehler++;
    }

    public static int anzahlErzeugteExemplare()
    {
        return exemplarzaehler;
    }
}
```

Klassenvariable

Auch hier gilt: Klassenvariablen sollten privat deklariert werden

Klassenmethode

Klassenmethoden

- Die öffentlichen Klassenmethoden bilden die Operationen eines Klassenobjektes

- Die Operationen eines Klassenobjektes sind für Klienten in der Punktnotation aufrufbar:

```
<Klassenname>.<Klassenoperation>(<aktuelle Parameter>);
```

```
class Kontenverwalter
{
    public void statusPruefen()
    {
        int anzahlKonten = Konto.anzahlErzeugteExemplare();
        ...
    }
}
```

Klassenoperationen als Dienstleistungen

- Statischen Methoden beziehen sich für die Dienstleistung nicht auf den Zustand des gerufenen Klassenobjekts, sondern **ausschließlich auf die übergebenen Parameter**

- Vorteil: Zum Abrufen dieser Dienstleistungen muss kein Exemplar einer Klasse erzeugt werden; das Klassenobjekt steht unmittelbar zur Verfügung

- Beispiele:
  - Die Klasse **Arrays** aus dem Paket **java.util**, die ausschließlich statische Methoden anbietet, mit denen Arrays manipuliert werden können (Arrays werden als Parameter übergeben)
  - Die Klassenoperation **arraycopy** in der Klasse **java.lang.System**. Sie bietet eine dritte Möglichkeit zum Kopieren von Array-Inhalten (neben dem expliziten Traversieren und **clone**)
  - Mathematische Funktionen in **java.lang.Math**.

Klassenoperation main

- Eine Klasse kann eine Klassenmethode mit einer ganz speziellen Signatur anbieten:

```
public static void main(String[] args)
```

- **Diese** Klassenoperation wird in der Laufzeitumgebung von Java gesondert behandelt (Schnittstelle zum Betriebssystem)
- **Einstiegspunkt für Java-Programme**: In dieser Methode werden üblicherweise die ersten Exemplare erzeugt, mit denen eine Java-Anwendung gestartet wird
- Interaktive Objekterzeugung ist eine Besonderheit von BlueJ geboten
- Andere IDEs bieten einen **Startknopf**, mit dem eine main-Methode aufgerufen wird

System.out erklärt

- Ausgaben auf die Konsole mit der Anweisung:

```
System.out.println("Hello World!");
```

- Diese ungewöhnliche Anweisung ist nun etwas besser erklärbar: Die Klasse `java.lang.System` verfügt über eine öffentliche Klassenkonstante **`out`**

- Diese Konstante ist vom Typ **`PrintStream`** und somit eine **konstante Referenz** auf ein Exemplar der Klasse **`java.io.Printstream`**

- Ein **`PrintStream`** ermöglicht mit seinen Operationen (u.a. **`println`**) die Ausgabe von Zeichenströmen

Initialisierung von Klassenobjekten

- Jede **Klasse** in Java definiert nur genau **ein Klassenobjekt**
- Dieses **Klassenobjekt** wird **automatisch erzeugt**, sobald eine Klasse in die Virtual Machine geladen wird
- **Keine** aufrufbaren **Konstruktoren** für Klassenobjekte
- In einer Klassendefinition können aber **Klassen-Initialisierer** angegeben werden, die nach dem Laden der Klasse ausgeführt werden

```
class Konto
{
    static {
        exemplarzaehler = 42;
        ...
    }
}
```

Klassenkonstanten

- Auch Konstanten (gekennzeichnet durch den Modifikator **`final`**) können mit dem Modifikator **`static`** deklariert werden
- Sie werden dadurch zu **Klassenkonstanten**
- Klassenkonstanten werden öffentlich (**`public`**) deklariert, wenn sie als globale Konstanten dienen sollen
- Beispiele:

```
public static final int TAGE_PRO_WOCHE = 7;
public static final float PI = 3.141592654f;
public static final int ANZAHL_SPALTEN = 80;
```

Hinweis zur Pragmatik: Fast immer sollten im Quelltext solche **symbolischen Konstanten** verwendet werden, anstatt an allen benutzenden Stellen jeweils das gewünschte Literal direkt anzugeben

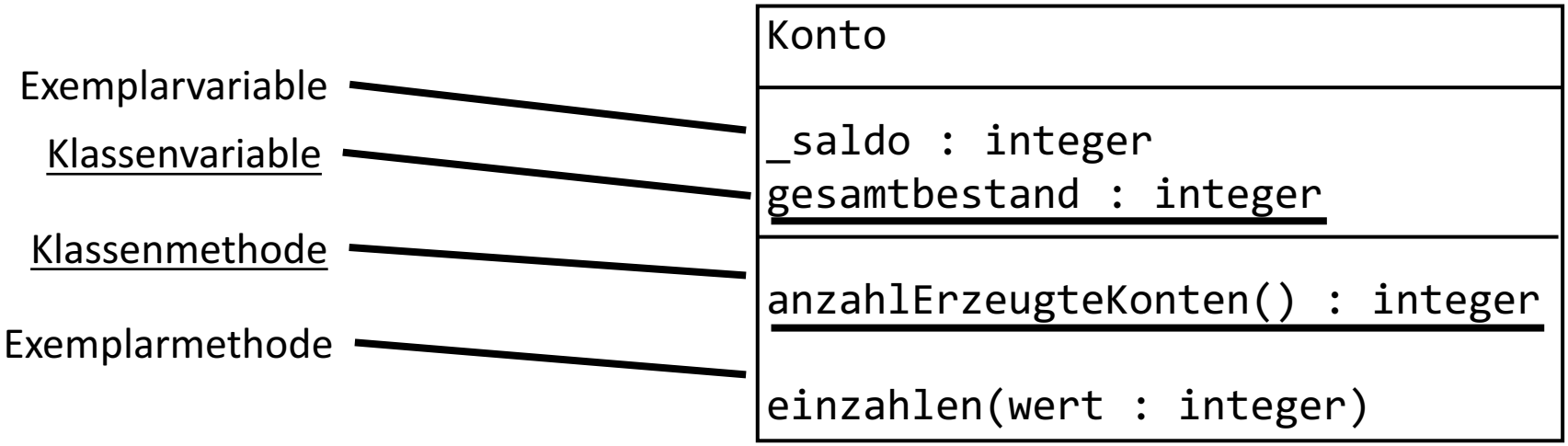
Nicht alle Objekte sind Exemplare

Nach der Einführung von Klassenobjekten können wir eine Unterscheidung zwischen **Exemplar** und **Objekt** für Java vornehmen:

- Alle Exemplare einer Klasse sind Objekte
- Auch eine Klasse ist ein Objekt, sie ist aber in Java nicht das Exemplar einer weiteren Klasse

- Exemplare werden explizit mit **new** erzeugt, während Klassen automatisch geladen und initialisiert werden, sobald sie benutzt werden

Klassenmethoden und -variablen in UML



Klassenvariablen und Klassenmethoden werden in den Klassen-Diagrammen der UML **unterstrichen**, um sie von Exemplarvariablen und -methoden zu unterscheiden

Zusammenfassung

1. In Java sind Klassen auch Objekte mit einem eigenen Zustand, der zur Laufzeit verändert werden kann. Die dazu notwendingen Klassenvariablen werden mit dem Schlüsselwort **`static`** deklariert.

2. Die Operationen eines Klassenobjektes werden mit Klassenmethoden realisiert, ebenfalls mit dem Schlüsselwort **`static`**.

3. Das Betriebsystem nutzt die **main-Methode** mit spezieller Signatur um ein Javaprogramm zu starten.

4. Öffentliche **Klassenkonstanten** (`public static final`) werden verwendet um globale Konstanten zu definieren.