# Java-Crashkurs

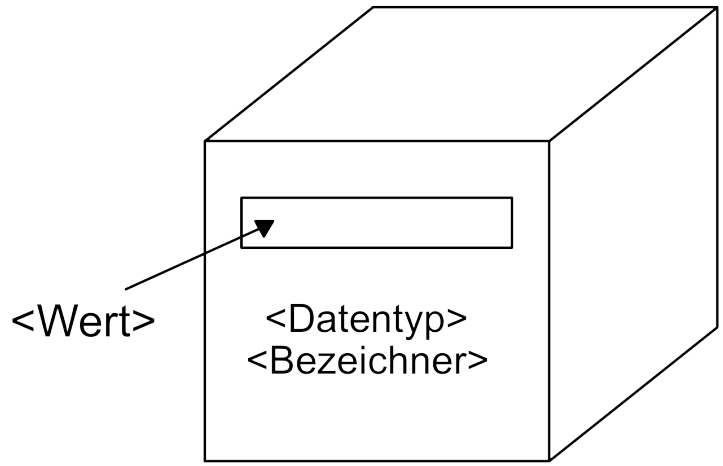
1. Datentypen & Operatoren | Benno Hölz, Matthias Ruf | 05.09.2023

sfz®
schülerforschungszentrum
südwürttemberg

# 1. Variablen

# Variable

Wie stellen wir uns eine Variable vor?




Anschauung bezieht sich auf Variablen von einem primitiven Datentypen

# Variablen

- Werte und Daten werden in Java in sogenannten Variablen gespeichert.
- Variablen müssen deklariert (erstellt) werden.
- optional kann ihnen ein Wert zugewiesen werden. (Initialisieren)
- Weist man einer Variable keinen Wert zu so setzt der Compiler diesen in bestimmten Kontexten auf einen Standardwert. (0,false,...)

# 2. Deklarieren und Initialisieren

# Deklarieren und Initialisieren

## Deklarieren

- erstellen einer Variable

```
1 //Reserviert einen Speicherplatz fuer einen Zahlenwert
2 <Datentyp> <Bezeichner>;
3 int zahl;
```

## Initialisieren

- **erste** Wertzuweisung

```
1 //Weist der Variable zahl den Wert 42 zu
2 <Bezeichner> = <Wert>;
3 zahl = 42;
```

# 3. Datentypen

# Datentypen

Wozu Datentypen?

- Klassifizierung von Variablen
- Eingrenzung des Wertebereichs
- Einfachere Interpretation bestimmter Operationen für den Compiler:
  - bspw. eine Multiplikation macht nur bei Zahlen Sinn, nicht für Wahrheitswerte

# Datentypen

**Was wollen wir speichern?**

# Datentypen

**Was wollen wir speichern?**

- Ganzzahlen
- Gleitkommazahlen
- Buchstaben
- Wörter/Zeichenketten
- Wahrheitswerte

# 4. primitive Datentypen

# primitive Datentypen

**Welche primitiven Datentypen stellt Java zur Verfügung?**

- Ganzzahlen
- Kommazahlen
- Buchstaben
- Wahrheitswerte

Für Zeichenketten gibt es andere Speichermethoden die wir auch noch kennen lernen werden. Man spricht von Zeigervariablen

# primitive Datentypen – Ganzzahlen

Für welche Daten verwenden wir welchen Datentyp?
**Ganzzahlen:**

- Byte
- Short
- Integer
- Long

# primitive Datentypen – Kommazahlen

Für welche Daten verwenden wir welchen Datentyp?
**Kommazahlen:**

- Float
- Double

# primitive Datentypen – Buchstaben/Zeichen

Für welche Daten verwenden wir welchen Datentyp?
**Buchstaben/Zeichen:**

- Char

# primitive Datentypen – Wahrheitswerte

Für welche Daten verwenden wir welchen Datentyp?
**Wahrheitswerte:**

- Boolean

# Ganzzahlen

Worin unterscheiden sich die Datentypen?

**Datentypen für Ganzzahlen in Java**

| Typ | Java | Wertebereich |
|---|---|---|
| **Byte** | byte | $-128$ bis $127$ |
| **Short** | short | $-32.768$ bis $32.767$ |
| **Integer** | int | $-2.147.483.648$ bis $2.147.483.647$ |
| **Long** | long | $-2^{63}$ bis $2^{63}-1$ |

Warum gibt es dann nicht nur Long?

# Kommazahlen – Loss of Precision

**Loss of Precision**

Achtung: Es gibt Gleitkommazahlen Zahlen im Dezimalsystem die im Binärsystem nicht genau dargestellt werden können.
Dies kann zu Rundungsfehlern führen!

**Beispiel**

z. B.: $33.55 * 100 = 3354.9999999999995$
(richtig wäre jedoch: $33.55 * 100 = 3355$)

Dieser Rundungsfehler sollte bedacht werden, da dieser zu Seiteneffekten führen kann.

# Wahrheitswerte - boolean

**Wahrheitswerte in Java**

- Java: boolean
- Wertebereich: true, false

# Buchstaben/Zeichen - char

**Buchstaben/Zeichen in Java**

- char Variablen werden mit dem ASCII-Code “repräsentiert“

# ASCII-Code

| dezimal | Zeichen | dezimal | Zeichen | dezimal | Zeichen | dezimal | Zeichen |
|---|---|---|---|---|---|---|---|
| 0 | [null] | 32 | [space] | 64 | @ | 96 | ‘ |
| 1 | [start of heading] | 33 | ! | 65 | A | 97 | a |
| 2 | [start of text] | 34 | ” | 66 | B | 98 | b |
| 3 | [end of text] | 35 | # | 67 | C | 99 | c |
| 4 | [end of transmission] | 36 | $ | 68 | D | 100 | d |
| 5 | [enquiry] | 37 | % | 69 | E | 101 | e |
| 6 | [acknowlege] | 38 | & | 70 | F | 102 | f |
| 7 | [bell] | 39 | ’ | 71 | G | 103 | g |
| 8 | [backspace] | 40 | ( | 72 | H | 104 | h |
| 9 | [horizontal tab] | 41 | ) | 73 | I | 105 | i |
| 10 | [line feed] | 42 | * | 74 | J | 106 | j |
| 11 | [vertical tab] | 43 | + | 75 | K | 107 | k |
| 12 | [form feed] | 44 | , | 76 | L | 108 | l |
| 13 | [carriage return] | 45 | - | 77 | M | 109 | m |
| 14 | [shift out] | 46 | . | 78 | N | 110 | n |
| 15 | [shift in] | 47 | / | 79 | O | 111 | o |
| 16 | [data link escape] | 48 | 0 | 80 | P | 112 | p |
| 17 | [device control 1] | 49 | 1 | 81 | Q | 113 | q |
| 18 | [device control 2] | 50 | 2 | 82 | R | 114 | r |
| 19 | [device control 3] | 51 | 3 | 83 | S | 115 | s |

# ASCII-Code

| dezimal | Zeichen | dezimal | Zeichen | dezimal | Zeichen | dezimal | Zeichen |
|---|---|---|---|---|---|---|---|
| 20 | [device control 4] | 52 | 4 | 84 | T | 116 | t |
| 21 | [negative acknowledge] | 53 | 5 | 85 | U | 117 | u |
| 22 | [synchronnous idle] | 54 | 6 | 86 | V | 118 | v |
| 23 | [end of trans. block] | 55 | 7 | 87 | W | 119 | w |
| 24 | [cancel] | 56 | 8 | 88 | X | 120 | x |
| 25 | [end of medium] | 57 | 9 | 89 | Y | 121 | y |
| 26 | [substitute] | 58 | : | 90 | Z | 122 | z |
| 27 | [escape] | 59 | ; | 91 | [ | 123 | { |
| 28 | [record separator] | 60 | < | 92 | \ | 124 | — |
| 29 | [file separator] | 61 | = | 93 | ] | 125 | } |
| 30 | [record separator] | 62 | > | 94 | ^ | 126 | ~ |
| 31 | [unit seperator] | 63 | ? | 95 | _ | 127 | [del] |

# Aus char mach int

Um aus der Char Variable c den Wert zu ermitteln muss der hinterlegte Wert manipuliert werden:

```
1 [...]
2 char c = '1';
3 int i = 41 + c; // 41 + 49
4 System.out.println(i);
5 [...]
```

Systemausgabe → 90

# Aus char mach int

Um aus der Char Variable c den Wert zu ermitteln muss der hinterlegte Wert manipuliert werden:

```
1 [...]
2 char c = '1';
3 int i = 41 + c - '0'; // 41 + 49 - 48
4 System.out.println(i);
5 [...]
```

Systemausgabe → 42

# 5. Operatoren auf primitive Datentypen

# Operatoren

**Welche Operatoren benötigen wir?**

# Operatoren

**Welche Operatoren benötigen wir?**

## Operatoren in Java

- Rechenoperatoren $(+, -, *, /, \%)$
- Vergleichsoperatoren $(==, !=, >, <, >=, <=)$
- Logikoperatoren $(\&\&, ||)$
- Zuweisungsoperatoren $(=, +=, -=, *=, /=)$

# Rechenoperatoren

## Beispiele für Addition, Subraktion und Multiplikation, sowie Modulo

```
1  //Deklarieren und Initalisieren
2  int operant1 = 6;
3  int operant2 = 5;
4  //Addition
5  int summe = operant1 + operant2;
6  //Subtraktion
7  int differenz = operant1 - operant2;
8  //Multiplikation
9  int produkt = operant1 * operant2;
10 //Modulo
11 int modulo = operant1 % operant2;
```

# Rechenoperatoren

## 2 Arten der Division

- Integer-Division / Division mit Rest:

```
1 int operant1 = 11;
2 int operant2 = 2;
3 int division = 11/2 // = 5
4 int rest = 11 % 2   //= 1
```

- Gleitkommazahlen-Division / normale Division:

```
1 float operant1 = 11;
2 float operant2 = 2;
3 float division = 11/2 // = 5.5
4 float rest = 11 % 2   //= 1.0
```

# Post-inkrement

Post-inkrement

```
1 i = 0;
2 System.out.println(i++);
3
4 // Ausgabe 0 Wert 1
5 System.out.println(i--);
6
7 // Ausgabe 1 Wert 0
```

(gleicher Code) :

```
1 i = 0;
2 System.out.println(i);
3 i = i + 1;
4 // Ausgabe 0 Wert 1
5 System.out.println(i);
6 i = i - 1;
7 // Ausgabe 1 Wert 0
```

# Pre-inkrement

Pre-inkrement:

```
1 i = 0;
2 System.out.println(++i);
3
4 // Ausgabe 1 Wert 1
5 System.out.println(--i);
6
7 // Ausgabe 0 Wert 0
```

(gleicher Code) :

```
1 i = 0;
2 i = i + 1;
3 System.out.println(i);
4 // Ausgabe 1 Wert 1
5 i = i - 1;
6 System.out.println(i);
7 // Ausgabe 0 Wert 0
```

# Logische Operatoren

## AND
## &&

```
1 boolean a = ?;
2 boolean b = ?;
3 boolean c = a && b;
```

| a | b | c |
|---|---|---|
| ✓ | ✓ | ✓ |
| ✓ | ✗ | ✗ |
| ✗ | ✓ | ✗ |
| ✗ | ✗ | ✗ |

✗= false, ✓= true

## OR
## ||

```
1 boolean a = ?;
2 boolean b = ?;
3 boolean c = a || b;
```

| a | b | c |
|---|---|---|
| ✓ | ✓ | ✓ |
| ✓ | ✗ | ✓ |
| ✗ | ✓ | ✓ |
| ✗ | ✗ | ✗ |

✗= false, ✓= true

# Vergleichsperatoren

## Gleich
==

```
1 boolean a = ?;
2 boolean b = ?;
3 boolean c = a == b;
```

| a | b | c |
|---|---|---|
| ✓ | ✓ | ✓ |
| ✓ | ✗ | ✗ |
| ✗ | ✓ | ✗ |
| ✗ | ✗ | ✓ |

✗= false, ✓= true

## Ungleich
!=

```
1 boolean a = ?;
2 boolean b = ?;
3 boolean c = a != b;
```

| a | b | c |
|---|---|---|
| ✓ | ✓ | ✗ |
| ✓ | ✗ | ✓ |
| ✗ | ✓ | ✓ |
| ✗ | ✗ | ✗ |

✗= false, ✓= true

# Vergleichsoperatoren

Vergleichsoperationen eignen sich nur für primitive Datentypen. (Für alle anderen existiert eine Methode siehe Kapitel 2 equals())

```
1  int a = 2;
2  int b = 2;
3  int c = 3;
4  boolean erg;
5  erg = a == b; //true
6  erg = a == c; //false
7  erg = a != c; //true
8  erg = a != b; //false
9  erg = a >= b; //true
10 erg = a >  b; //false
11 erg = a <= b; //true
12 erg = a <  b; //false
```

# Zuweisungsoperatoren

Der Zuweisungsoperator = eignet sich für jeden Datentyp, die anderen Zuweisungsoperatoren nur für Zahlenwerte.

```
1 int a = 2;
2 int b = a;  // b = a; also 2
3 b += a; // b = b + a; also 4
4 b *= a; // b = b * a; also 8
5 b /= a; // b = b / a; also 4
6 b -= a; // b = b - a; also 2
```

# 6. Typecasting

# Typecasting

**Impliziter Typecast**

Rechnet man mit Variablen unterschiedlicher Datentypen, so werden diese automatisch in den Typ mit dem größten Wertebereich umgewandelt (**impliziter Typecast**).

```
1 int i = 6;
2 float f = 4;
3 System.out.println(i + f); // 10.0
```

# Typcasting

Beim implizienten Typcasting wird entlang folgendes Pfades gecastet:

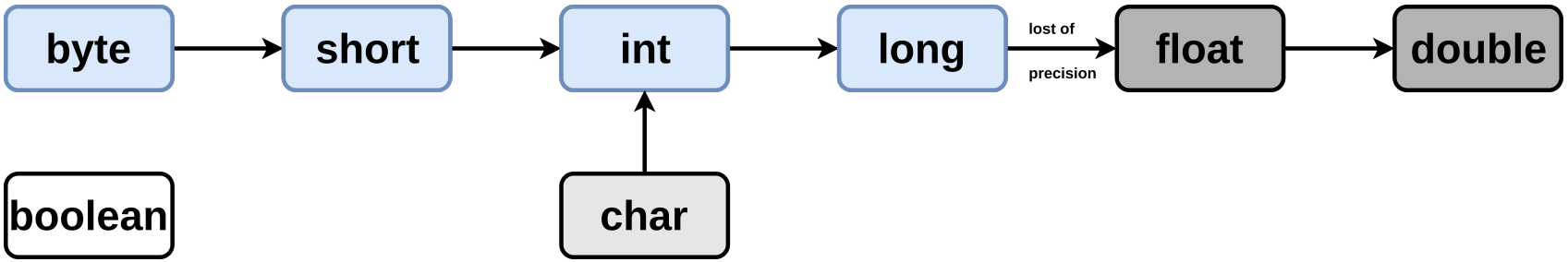



Abbildung: implizites Typcasting

`boolean` als primitiver Datentyp kann nicht gecastet werden. (auch nicht explizit)

# Typecasting

## Expliziter Typecast

Möchte man jedoch mit einem kleineren Typ weiter rechnen, so ist ein **expliziter Typecast** nötig.
Dies wird mit einem vorangestellten (< *Zieldatentyp* >) erreicht:

```
1 int i = 3;
2 float f = 4.3f;
3 System.out.println(i + f); // 7.3
4 System.out.println(i + (int) f); // 7
```

Dabei gehen ggf. Nachkommastellen (etc.) verloren!

# 7. Strings

# Strings

## Strings in Java – Eigenschaften

- Strings sind Zeichenketten

# Strings

## Strings in Java – Eigenschaften

- Strings sind Zeichenketten
- Strings gelten **nicht** als primitiver Datentyp

# Strings

## Strings in Java – Eigenschaften

- Strings sind Zeichenketten
- Strings gelten **nicht** als primitiver Datentyp
  - Sie haben keinen festen Wertebereich

# Strings

## Strings in Java – Eigenschaften

- Strings sind Zeichenketten
- Strings gelten **nicht** als primitiver Datentyp
  - Sie haben keinen festen Wertebereich
  - Ihre Länge kann dynamisch verändert werden

# Strings

**Strings in Java – Eigenschaften**

- Strings sind Zeichenketten
- Strings gelten **nicht** als primitiver Datentyp
  - Sie haben keinen festen Wertebereich
  - Ihre Länge kann dynamisch verändert werden
- Alle Datentypen (primitiv oder nicht) können in einen String konvertiert werden.

# Erzeugen von Strings

## Erzeugen von Strings

Für uns sind zwei Arten relevant:

- Konstruktor

```
1 String str = new String("Hallo Welt");
```

Der Konstruktor wird uns beim Thema Objektorientierung nochmals begegnen, ist aber in Bezug auf Strings weniger relevant

# Erzeugen von Strings

## Erzeugen von Strings

Für uns sind zwei Arten relevant:

- Konstruktor

```
1 String str = new String("Hallo Welt");
```

- oder einfacher nur mit Anführungszeichen

```
1 String str = "Hallo Welt";
```

Der Konstruktor wird uns beim Thema Objektorientierung nochmals begegnen, ist aber in Bezug auf Strings weniger relevant

# Umwandeln in Strings

Es gibt auch hier verschiedene Arten:

- toString() - Methode

```
1 int a = 5;
2 String str = a.toString();
```

# Umwandeln in Strings

Es gibt auch hier verschiedene Arten:

- toString() - Methode

```
1 int a = 5;
2 String str = a.toString();
```

- Einen leeren String voranstellen

```
1 int a = 5;
2 String str = "" + a;
```

# Anhängen an einen String

Wir verwenden die bekannten Vorangehensweisen:

```
byte b = 1;
short s = 2;
int i = 3;
long l = 4;
float f = 5.0f;
double d = 6.0d;
String str = "Zahlen: " + b + s + l + f + d;
// Zahlen: 12345.06.0
```

Achtung Seiteneffekte:

```
str = b + s + " sind Zahlen.";
// 3 sind Zahlen.
```

oder

```
str = "Summe aus 2 und 1 = " + (b + s);
// Summe aus 2 und 1 = 3
```

# Vergleichen von Strings

- Wenn String mit **new** erstellt wird: Vergleich über **== liefert nicht den inhaltlichen Vergleich**, sondern nur den Vergleich der Referenzen (Speicherort).

```
1 String a = new String("String!");
2 String b = new String("String!");
3 String c = new String("String_");
4
5 System.out.println(a == a); // true
6 System.out.println(a == b); // false
7 System.out.println(a == c); // false
8 System.out.println(b == b); // true
9 System.out.println("String!" == a ); // false
```

⇒ **besser: equals-Methode**

# Vergleichen von Strings

Die **equals-Methode**:

- liefert **inhaltlichen Vergleich** von Strings

```
1 String a = "String!";
2 String b = "String!";
3 String c = "String_";
4
5 System.out.println(a.equals(a)); // true
6 System.out.println(a.equals(b)); // true
7 System.out.println(a.equals(c)); // false
8 System.out.println(b.equals(b)); // true
9 System.out.println("String!".equals(a)); // true
```

⇒ Vergleichen von Strings? equals-Methode benutzen.

# String als Klasse

Was wollen wir mit Strings machen?

- Inhaltlich vergleichen [⇒equals-Methode]

# String als Klasse

Was wollen wir mit Strings machen?

- Inhaltlich vergleichen [⇒equals-Methode]
- Länge wissen [⇒length-Methode]

# String als Klasse

Was wollen wir mit Strings machen?

- Inhaltlich vergleichen [⇒equals-Methode]
- Länge wissen [⇒length-Methode]
- Zeichen an einer best. Stelle [⇒charAt-Methode]

# String als Klasse

- Gibt die Länge des Strings als Integer zurück.

```
1 String str = "12345";
2 int length = str.length();        // 5
3 str = "abcdefghijklmnopqrstuvwxyz";
4 length = str.length();            // 26
```

oder auch direkt ohne Bezeichner:

```
1 int length = "123".length();      // 3
```

# String als Klasse

str.charAt(int index);

- Gibt den Char zurück der am Index “index“ steht.
- Index beginnt bei 0 und endet bei str.length() -1

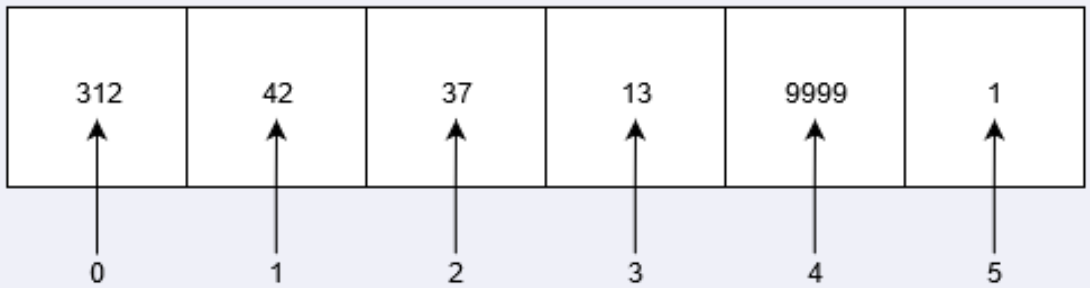
```
1              // 01234
2 String str = abcde;
3 char a = str.charAt(0);
4 char b = str.charAt(1);
5 char c = str.charAt(2);
6 char d = str.charAt(3);
7 char e = str.charAt(4);
8 int length = str.length(); // 5
```

# 8. Arrays

# Was ist ein Array?

**Array**

- engl. für *Feld*
- Datenstruktur
- Zugriff mithilfe von Indizes (beginnend bei 0)



# Deklarieren und Initialisieren

- **Deklaration:**

```
1 <Datentyp>[] <Bezeichner> = new <Datentyp>[<Laenge>];
2 int[] a = new int[6]; // Integer-Array der Laenge 6
```

- **Initialisierung:**

```
1 <Bezeichner>[<Index>] = <Wert>;
2 a[2] = 37; // Element 2 mit Wert 37 belegen
```

- Zugriff auf Elemente:

```
1 <Arrayname>[<Index>];
2 System.out.println(a[2]); // 37
```

# Deklarieren und Initialisieren

- Direkte Initialisierung:

```
1 <Datentyp>[] <Bezeichner> = {<Wert>,<Wert>,<...>};
2 int[] a = {312, 42, 37, 13, 9999, 1};
```

# Länge eines Arrays

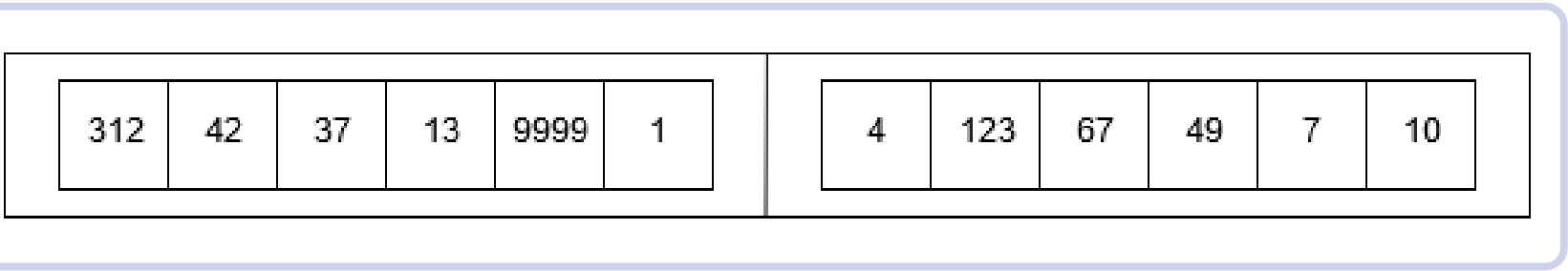
- Attribut `length` speichert die Länge eines Arrays

```
1 int[] a = new int[3];
2 System.out.println(a.length); // 3
```

↔ Strings: Bestimmung der Länge mit `length()` (→ Methode!)

# Mehrdimensionale Arrays

- bisher: eindimensionale Arrays
- auch mehrere Dimensionen möglich
- z. B. zweidimensionales Array:



# Mehrdimensionale Arrays

- Deklaration:

```
1 <Typ>[][] <Bez.> = new <Typ>[<Laenge 1>][<Laenge 2>];
2 int[][] a2 = new int[2][6];
```

- z. B. für Tabellen, Matrizen, Datenbanken, ...

# Typische Operationen

## Typische Operationen für Arrays

- `for`-Schleife
- `forEach`-Schleife:

```java
int[] arr = {1, 4, 9, 42};
for(int i: arr){
        System.out.println(i);
}

// Ausgabe:
// 1
// 4
// 9
// 42
```

Mehr dazu im Kapitel zu Schleifen

# 9. Zusammenfassung

# Zusammenfassung – primitive Datentypen

- Es gibt verschiedene Datentypen
  - `byte`, `short`, `int`, `long` für Ganzzahlen
  - `float`, `double` für Gleitkommazahlen
  - `boolean` für Logische Ausdrücke
  - `char` für Einzelzeichen
- Deklarieren und Initalisieren

```
1 int a;              //deklalieren
2 a = 42;             // initalisieren
3 float f = 42.0f; // beides zusammen
```

- Es gibt verschiedene Operatoren
  - Rechenoperatoren `+,-,*,/,%`
  - Vergleichsoperatoren `<, >, >=, <=, ==, !=`
  - Logikoperatoren `||, &&, !`
  - Zuweisungsoperatoren `=, *=, +=, -=, /=`

# Zusammenfassung – Strings

- Zeichenketten

```
1 String str = "Hallo Welt";
```

- Index beginnt bei 0

```
1             //  0123456789
2 String str = "Hallo Welt";
```

- Länge mit der Methode: str.length();

```
1 System.out.println(str.length()); // -> 10
```

- Zeichen an einer Stelle mit: str.charAt(int index);

```
1 System.out.println(str.charAt(1)); // -> a
```

- Vergleichen mit str1.equals(String str2);

```
1 System.out.println(str == "Hallo Welt"); // false
2 System.out.println(str.equals("Hallo Welt")); // true
```

# Zusammenfassung – Arrays

- Deklaration:

```
int[] a = new int[6]; // 6-stelliges int-Array
```

- Initialisierung:

```
a[2] = 42; // 2. Element hat Wert 42
int[] a = {1, 4, 7, 13, 2}; // direkt
```

- mehrdimensionale Arrays

```
int[][] arr = new int[1][2];
arr[0][1] = 4;
```

- `for`-Schleife / `forEach`-Schleife

```
System.out.print("Array:");
for(int i : a){
        System.out.print(" " + i + ",");
}
System.out.println();
```