Spickzettel Java · Grundlagen der Programmierung

Programm · Daten · Rechnen · Text · Ablauf · Methoden · Arrays · Hilfe

Holger Sebastião · www.sebastiao.org
Version 1.8 · 2026-10-25

Programmgerüst

```
// Datei: HelloWorld.java
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

Dateiname = Klassenname (`HelloWorld.java`). Das Programm startet in `main`.

Jede Anweisung endet mit `;`. Groß- und Kleinschreibung zählt: `summe` ≠ `Summe`.

Ausgabe

```
System.out.println("Hallo"); // + neue Zeile
System.out.print("Hallo");   // ohne Umbruch
System.out.println("Preis: " + preis);
System.out.printf("%.2f €%n", preis);
// → 3,50 €  (2 Nachkommastellen)
```

`%.2f` Kommazahl mit 2 Stellen, `%d` Ganzzahl, `%s` Text, `%n` neue Zeile.

! `"Summe: " + 3 + 4` → `Summe: 34`
Klammern setzen: `"Summe: " + (3 + 4)` → `Summe: 7`

Kommentare & Sonderzeichen

```
// einzeilig
/* mehrzeilig */
/** Beschreibung einer Methode (Javadoc) */
```

`\n`	neue Zeile	`\"`	Anführungszeichen
`\t`	Tabulator	`\\`	Backslash

Variablen & Konstanten

```
int alter;          // deklarieren
alter = 17;         // Wert zuweisen
int punkte = 0;     // beides in einer Zeile
punkte = punkte + 10;
final double MWST = 0.19;  // Konstante
```

`=` ist die **Zuweisung**: rechte Seite ausrechnen, links speichern.

Konstante mit `final`: Wert ist danach nicht mehr änderbar. Im Struktogramm steht `const`. `Math.PI` gibt es schon.

Eingabe — Scanner

```
import java.util.Scanner;  // ganz oben
Scanner eingabe = new Scanner(System.in);

String wort  = eingabe.next();     // 1 Wort
String zeile = eingabe.nextLine(); // Zeile
int alter    = eingabe.nextInt();
double preis = eingabe.nextDouble();
```

! Nach `nextInt()` oder `nextDouble()` bleibt das Enter übrig. Ein folgendes `nextLine()` liefert dann `""`. Abhilfe: einmal extra `eingabe.nextLine();`

! `nextDouble()` erwartet auf deutschem System ein **Komma**: `3,5`

`eingabe.close();` erst am Programmende, danach liest kein Scanner mehr.

Datentypen

TYP	BIT	INHALT	BEISPIEL
`byte`	8	−128 … 127	
`short`	16	−32 768 … 32 767	
`int`	**32**	**±2,1 Milliarden**	`int anzahl = 5;`
`long`	64	±9,2 Trillionen	`long n = 5000000000L;`
`float`	32	≈ 7 Stellen genau	`float f = 2.5f;`
`double`	**64**	**≈ 15 Stellen genau**	`double p = 3.99;`
`char`	16	ein Zeichen	`char c = 'A';`
`boolean`		**true / false**	`boolean ok = true;`
`String`		**Text (Klasse)**	`String s = "Ada";`

Fett = Standard im Unterricht. Kommazahlen mit **Punkt**: `3.99`. `'A'` ist ein `char`, `"A"` ein `String`. String ist eine Klasse, deshalb groß.

Namen

`KinoKasse`	Klassen: UpperCamelCase
`preisMitSteuer`	Variablen, Methoden: lowerCamelCase
`MAX_WERT`	Konstanten: GROSS_MIT_UNTERSTRICH

Keine Leerzeichen, nicht mit Ziffer beginnen, keine Schlüsselwörter wie `int` oder `class`. Umlaute vermeiden.

Sprechende Namen: `anzahlSchueler` statt `a`.

Rechnen

`+` `-` `*` `/` `%` (Rest). Punkt vor Strich, Klammern zuerst.

```
7 / 2        // 3    int / int: Rest weg!
7.0 / 2      // 3.5
7 % 2        // 1    Rest (Modulo)
zahl % 2 == 0 // true, wenn zahl gerade
```

! Sind beide Zahlen `int`, ist auch das Ergebnis `int`: `1 / 2` → `0`.

Kurzschreibweisen — nicht empfohlen

```
punkte = punkte + 10;  // kurz: punkte += 10;
preis = preis * 2;     // kurz: preis *= 2;
i = i + 1;             // kurz: i++;
i = i - 1;             // kurz: i--;
```

Schreib die Langform. Die verstehst du auch in zwei Wochen noch. Die Kurzform musst du nur lesen können, sie steht oft in fremdem Code. Ausnahme: `i++` im Kopf der for-Schleife.

Math — kein import nötig

```
Math.sqrt(16)     // 4.0  Wurzel
Math.pow(2, 3)    // 8.0  2 hoch 3
Math.abs(-5)      // 5    Betrag
Math.max(3, 8)    // 8    (min ebenso)
Math.round(2.5)   // 3    runden
```

Zufallszahlen

```
double z = Math.random(); // 0.0 bis unter 1.0
// 0 bis 9:
int ziffer = (int) (Math.random() * 10);
// 1 bis 6:
int wuerfel = (int) (Math.random() * 6) + 1;
```

Allgemein:
`(int) (Math.random() * (max - min + 1)) + min`

Typumwandlung — Casting

```
double d = 5;              // 5.0 automatisch
int ganz = (int) 3.7;      // 3 abgeschnitten!
int r = (int) Math.round(3.7); // 4 gerundet
double q = (double) 7 / 2; // 3.5
char c = (char) 65;        // 'A'
int code = (int) 'A';      // 65
```

Klein → groß automatisch, groß → klein nur mit `(Typ)`.

Text ↔ Zahl

```
int n = Integer.parseInt("42");
double d = Double.parseDouble("3.14");
boolean b = Boolean.parseBoolean("true");
String s = String.valueOf(42); // oder 42 + ""
```

`parseDouble` erwartet einen **Punkt**. Ist der Text keine Zahl (`"4x"`) → `NumberFormatException`.

Zeichenketten — String text = "Hallo";

`text.length()`	`5`
`text.charAt(0)`	`'H'` (ab 0 zählen)
`text.substring(1, 3)`	`"al"` (Ende zählt nicht mit)
`text.indexOf("l")`	`2` (nicht da: `-1`)
`text.contains("all")`	`true`
`text.equals("Hallo")`	`true`
`text.equalsIgnoreCase("hallo")`	`true`
`text.toUpperCase()`	`"HALLO"` (`toLowerCase` ebenso)
`text.replace("l", "x")`	`"Haxxo"` (alle)
`text + " Welt"`	`"Hallo Welt"`
`" Hi ".strip()`	`"Hi"` (nur am Rand)

`trim()` wie `strip()`, kennt aber nicht alle Leerzeichen. `stripLeading()` nur links, `stripTrailing()` nur rechts.

! Strings ändern sich nie selbst: `text.toUpperCase();` allein bewirkt nichts. Richtig: `text = text.toUpperCase();`

! Texte mit `equals` vergleichen, nie mit `==`.

```
for (int i = 0; i < text.length(); i++) {
  System.out.println(text.charAt(i));
}
```

Klammern & import

`{ }`	Block: Klasse, Methode, if, Schleife
`( )`	Bedingung, Parameter, Cast, Rechnen
`[ ]`	Arrays

`import java.util.Scanner;` macht eine Klasse aus einem anderen Paket mit ihrem Kurznamen nutzbar. `String`, `Math` und `System` brauchen kein import.

Spickzettel Java · Grundlagen der Programmierung

Programm · Daten · Rechnen · Text · Ablauf · Methoden · Arrays · Hilfe

Holger Sebastião · www.sebastiao.org
Version 1.8 · 2026-10-25

Bedingungen

Eine Bedingung ergibt immer `true` oder `false`.

`==`	gleich	`!=`	ungleich
`<`	kleiner	`>`	größer
`<=`	kleiner oder gleich	`>=`	größer oder gleich
`&&`	UND: beide true	`||`	ODER: mind. eine
`!`	NICHT: kehrt um, `!fertig`		

```
boolean jugendlich = alter >= 14 && alter < 18;
```

- `=` weist zu, `==` vergleicht.
- `1 <= x <= 6` geht nicht. Richtig: `x >= 1 && x <= 6`
- Texte: `name.equals("Ada")`, nie `name == "Ada"`

Verzweigung `if`

```
if (note <= 4) {
  System.out.println("bestanden");
} else {
  System.out.println("nicht bestanden");
}
```

MEHRERE FÄLLE

```
if (punkte >= 90) {
  stufe = "sehr gut";
} else if (punkte >= 50) {
  stufe = "bestanden";
} else {
  stufe = "nicht bestanden";
}
```

Geprüft wird von oben nach unten, der erste passende Zweig wird ausgeführt. `else` darf fehlen.

- Kein `;` hinter `if (…)`, sonst gilt die Bedingung für nichts.

Pseudocode

```
LIES zahl
WENN zahl > 0 DANN
    GIB "positiv" AUS
SONST
    GIB "nicht positiv" AUS
```

Fallauswahl `switch`

MODERN, AB JAVA 14

```
switch (tag) {
  case 1 -> text = "Montag";
  case 2 -> text = "Dienstag";
  case 6, 7 -> text = "Wochenende";
  default -> text = "anderer Tag";
}
```

Kein `break` nötig. Mehrere Anweisungen: `case 1 -> { … }`

KLASSISCH

```
switch (tag) {
  case 6: case 7:
    text = "Wochenende";
    break;
  default:
    text = "anderer Tag";
}
```

- Ohne `break` läuft das Programm in den nächsten `case` weiter.

Geht mit `int`, `char`, `String`, nicht mit `double` oder `boolean`.

Schleife, kopfgesteuert `while`

```
int i = 0;
while (i < 5) {
  System.out.println(i);
  i = i + 1;
}
```

Prüft **vor** jedem Durchlauf. Ist die Bedingung gleich falsch, läuft sie kein einziges Mal.

- Fehlt `i = i + 1;`, wird die Bedingung nie falsch: Endlosschleife.

Schleife, fußgesteuert `do-while`

```
int zahl;
do {
  System.out.print("Zahl von 1 bis 6: ");
  zahl = eingabe.nextInt();
} while (zahl < 1 || zahl > 6);
```

Prüft **nach** jedem Durchlauf, läuft also mindestens einmal. Typisch: Eingabe wiederholen, bis sie stimmt. Am Ende steht ein `;`.

Zählschleife `for`

```
// for (Start; Bedingung; Änderung)
for (int i = 0; i < 5; i++) {
  System.out.println(i);  // 0 1 2 3 4
}
for (int i = 10; i > 0; i--) { … } // rückwärts
```

Zählvariable heißt `i`, verschachtelt außen `i`, innen `j`.

`break;` verlässt die innerste Schleife sofort. `continue;` überspringt den Rest dieses Durchlaufs.

Arrays

```
int[] zahlen = new int[5];  // 5 Fächer, alle 0
int[] noten = {2, 1, 3};    // mit Werten
noten[0] = 4;               // erstes Fach
int anzahl = noten.length;  // 3, ohne ()
```

Index von `0` bis `length - 1`. `noten[3]` → `ArrayIndexOutOfBoundsException`.

```
for (int i = 0; i < noten.length; i++) {
  System.out.println(noten[i]);
}
System.out.println(Arrays.toString(noten));
// → [4, 1, 3]
// braucht: import java.util.Arrays;
```

- `println(noten)` zeigt nicht die Werte, sondern nur so etwas wie `[I@1b6d3586`.

2D-Array `Tabelle`

```
int[][] sitze = new int[3][4];
// 3 Zeilen, 4 Spalten
sitze[1][2] = 1;   // [Zeile][Spalte]
sitze.length       // 3 Zeilen
sitze[0].length    // 4 Spalten

for (int i = 0; i < sitze.length; i++) {
  for (int j = 0; j < sitze[i].length; j++) {
    System.out.print(sitze[i][j] + " ");
  }
  System.out.println();
}
```

Methoden `Unterprogramme`

Kopf: `public static` **Rückgabetyp Name(Parameter)**

PROZEDUR, OHNE RÜCKGABEWERT

```
public static void begruesse(String name) {
  System.out.println("Hallo " + name);
}
begruesse("Ada");              // Aufruf
```

FUNKTION, MIT RÜCKGABEWERT

```
public static int verdopple(int zahl) {
  return zahl * 2;
}
int ergebnis = verdopple(5);  // 10
```

`return` gibt den Wert zurück und beendet die Methode. Variablen in einer Methode gelten nur dort.

Parameterübergabe

Zahlen, char, boolean: Die Methode bekommt eine Kopie des Werts. Der Aufrufer merkt keine Änderung (Wertübergabe).

Arrays: Die Methode bekommt einen Verweis auf dieselben Fächer. Änderungen an den Fächern sieht auch der Aufrufer (Referenzübergabe).

```
static void nullen(int[] a) { a[0] = 0; }
int[] z = {5, 6};
nullen(z);          // z[0] ist jetzt 0
```

Fehlermeldungen verstehen

BEIM ÜBERSETZEN

`';' expected`	Semikolon fehlt, oft eine Zeile darüber
`cannot find symbol`	Tippfehler oder nicht deklariert
`incompatible types`	Typ passt nicht: `(int)` oder `parseInt`
`missing return statement`	nicht jeder Weg endet mit `return`
`reached end of file`	eine `}` fehlt

BEIM AUSFÜHREN

`…IndexOutOfBounds…`	Index zu groß oder kleiner 0
`InputMismatchException`	Scanner bekam den falschen Typ
`NumberFormatException`	Text ist keine Zahl
`/ by zero`	Ganzzahl durch 0 geteilt

Java API: docs.oracle.com/en/java/javase/21/docs/api · Struktogramme: visucode.de