# Programmierkurs Java

## UE 4 - Datentypen

Dr.-Ing. Dietrich Boles

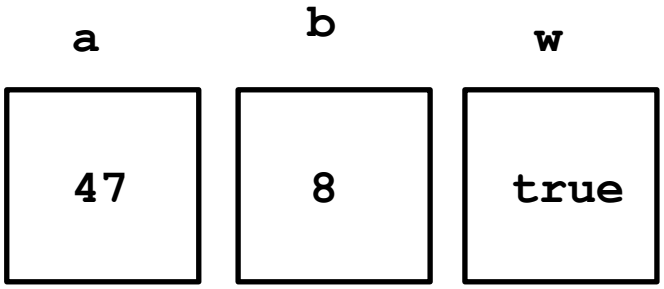
# Gliederung

- Computer Speicher
- Datentypen
- Variablen
- Literale
- Character
- Ausdrücke
- Operatoren
- Zeichenketten
- Eingabe mit Klasse IO
- Zusammenfassung

# Computer Speicher

```
int a = 47;
int b = 8;
boolean w = true;
```

Programm




Logische Representation

Physische Representation

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

Bit = 0 oder 1  1 byte = 8 bits

# Binärzahlen (1)

| Binärzahl | Dezimalzahl |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 10 | 2 |
| 11 | 3 |
| 100 | 4 |
| 101 | 5 |
| 110 | 6 |
| 111 | 7 |
| ... | |

**Dezimal:** $347 = 3 * 10^2 + 4 * 10^1 + 7 * 10^0$

**Binär:** $1101_2 = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 13_{10}$

# Binärzahlen (2)

**Addition:**

```
  11010    (26)
+ 10111    (23)
 ------
 110001    (49)
```

**Umwandlung:**

```
26 / 2 = 13   R 0
13 / 2 = 6    R 1
 6 / 2 = 3    R 0
 3 / 2 = 1    R 1
 1 / 2 = 0    R 1
```

# Hexedezimalzahlen (1)

| Hexadezimal | Dezimal |
|---|---|
| 0 | 0 |
| 1 | 1 |
| ... | |
| 9 | 9 |
| A | 10 |
| B | 11 |
| C | 12 |
| D | 13 |
| E | 14 |
| F | 15 |
| 10 | 16 |
| ... | |

$$2BF3_{16} = 2 * 16^3 + 11 * 16^2 + 15 * 16^1 + 3 * 16^0 = 11251_{10}$$

# Hexadezimalzahlen (2)

**Addition:**

```
      1F    (31)
+     11    (17)
 ------
      30    (48)
```

**Umwandlung:**

```
439 / 16 = 27 R  7
 27 / 16 =  1 R  B (11)
  1 / 16 =  0 R  1
```

# Datentypen (1)

- Daten haben unterschiedliche Eigenschaften:
  - Ganze Zahlen: 2, 4711, -45, ...
  - Reelle Zahlen: -3.4, -56.789678, -3.4e-10, ...
  - Buchstaben: 'a', 'b', ...
  - Zeichenketten: "hello world!", ...
  - ...
- Unterschiedliche Daten benötigen unterschiedlich viel Platz
  - boolesche Werte: 1 Bit
  - ASCII-Zeichen: 7 Bit
  - ...
- Unsinnige Operationen: `3 - "hello"`
- Datentyp:
  - Wertebereich
  - Operationen auf Werten des Wertebereichs

# Datentypen (2)

Einfache Datentypen in Java:

| | |
|---|---|
| **`boolean`** | entweder `true` oder `false` |
| **`char`** | 16-Bit-Unicode |
| **`byte`** | 8-Bit-Integer, vorzeichenbehaftetes 2er-Komplement |
| **`short`** | 16-Bit-Integer, vorzeichenbehaftetes 2er-Komplement |
| **`int`** | 32-Bit-Integer, vorzeichenbehaftetes 2er-Komplement |
| **`long`** | 64-Bit-Integer, vorzeichenbehaftetes 2er-Komplement |
| **`float`** | 32-Bit-Gleitkommazahl (IEEE 754-1985) |
| **`double`** | 64-Bit-Gleitkommazahl (IEEE 754-1985) |

# Datentypen (3)

**Zweierkomplement**:

`0111 1111 = 127` Vorzeichenbit

`1000 0000 = -128` Vorzeichenbit

Wertebereiche beschränkt: `int`: $[-2^{31} \ldots 2^{31}-1]$

```
int max = 2147483647; max = max + 1; max: -2147483648
```

Grund: `011...111 + 1 = 100...000`

**IEEE 754**:

`float`:

1 Vorzeichenbit

8 Bits für Exponenten

23 Bits für Mantisse

`double`:

1 Vorzeichenbit

11 Bits für Exponent

52 Bits für Mantisse

# Datentypen (4)

| | Default | Kleinster Wert | Größter Wert |
|---|---|---|---|
| boolean | false | N.A. | N.A. |
| char | '\u0000' | '\u0000' | '\uFFFF' |
| byte | 0 | -128 | 127 |
| short | 0 | -32768 | 32767 |
| int | 0 | -2147483648 | 2147483647 |
| long | 0L | -9223372036854775808 | 9223372036854775807 |
| float | 0.0F | +-3.40282347E+38 | +-1.40239846E-45 |
| double | 0.0 | +-1.79769313486231570E+308 | +-4.94065645841246544E-324 |

# Variablen / Definition

```
<Variablendefinition> ::= <Datentyp>
                          <Bezeichner> "=" <Ausdruck>
                          {"," <Bezeichner> "=" <Ausdruck>}
                          ";"
```

```
int anzahl = 0;
long grosseZahl = 1234567891234567L;

float pi = 3.1415F;
double nochEine = 4.5, nochZwei = 2.3;

char eins = '1', zwei = '2', drei = '3';
```

# Variablen / Initialisierung

- Initialisierung von Variablen
  - Festlegung eines anfänglichen Wertes
  - mit Hilfe von Literalen bzw. Ausdrücken

# Literale (1)

Literale: Typ-spezifische Konstanten

- `boolean`:

  **`true`**, **`false`**

- `int`:

  Zeichenketten aus

  – dezimalen **`29`**
  – oktalen (führende 0): **`035`**
  – hexadezimalen Ziffern (führendes 0x): **`0x1D`**

- `long`:

  – `int`-Literal mit nachgestelltem `l` oder `L`: **`29L`** **`43211`**

# Literale (2)

- `double:`

  Dezimalzahlen mit Dezimalpunkt und optionalem Exponent:

  **`18.`** **`1.8e1`** **`.18E-2`**

- `float:`

  `double`-Literal mit nachgestelltem `f` oder `F`:

  **`18.0f`** **`1.8e1F`**

- `char:`

  Zeichen zwischen zwei Apostrophen: **`'a'`** **`'A'`** **`'2'`**

  oder Sonderzeichen mit Escape-Sequenz: **`'\n'`** **`'\''`** **`'\\'`**

  **oder Unicode-Zeichen** **`'\u00A9'`** **`(= ©)`**

- *`Zeichenketten`*`:`

  Zeichen zwischen zwei Anführungszeichen: **`"hello\nworld"`**

# Character /ASCII-Tabelle

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DEL |

Source: www.LookupTables.com

# Ausdrücke / Definition

- Ausdruck:
  - Verarbeitungsvorschrift, deren Ausführung einen **Wert** eines bestimmten Typs liefert
  - entsteht, indem **Operanden** mit **Operatoren** verknüpft werden
  - Operanden müssen **typkonform** sein!

```
<Ausdruck> ::=   <Literal>
               | <Variablenname>
               | <Funktionsaufruf>
               | "(" <Ausdruck> ")"
               | <unär-Op> <Ausdruck>
               | <Ausdruck> <binär-Op> <Ausdruck>
               | <Ausdruck> <ternär-Op1>
                 <Ausdruck> <ternär-Op2>
                 <Ausdruck>
```

# Operatoren / Übersicht

- Ganzzahl-Arithmetik (int, short, long): `+, -, *, /, %`
- Gleitkomma-Arithmetik (float, double): `+, -, *, /, %`
- Boolesche Arithmetik (boolean): `&, |, ^, &&, ||, !`
- Vergleichsoperatoren: `==, !=, <, >, <=, >=`
- Zuweisungsoperatoren: `=, +=, -=, *=, /=, %=`
- Inkrement-Operator: `++`
- Dekrement-Operator: `--`
- Bit-Operatoren: `<<, >>, &, |, ~, ^, ...`
- Spezielle Operatoren: `?:, (type)`

# Operatoren / Ganzzahl-Arithmetik

- Operanden vom Typ `int`, `short` oder `long`
- gelieferter Wert vom Typ `int`, `short` bzw. `long`

| | | |
|---|---|---|
| **+**: | positives Vorzeichen | `+9876` |
| **-**: | negatives Vorzeichen | `-2345` |
| **+**: | Addition | `12 + 98 (= 110)` |
| **-**: | Subtraktion | `12 - 98 (= -86)` |
| *****: | Multiplikation | `6 * 5 (= 30)` |
| **/**: | Ganzzahl-Division | `7 / 3 (= 2)` |
| **%**: | Restbildung (modulo) | `7 % 3 (= 1)` |

# Operatoren / Gleitkomma-Arithmetik

- Operanden vom Typ `float` oder `double`
- gelieferter Wert vom Typ `float` oder `double`

| | | |
|---|---|---|
| **+**: | positives Vorzeichen | `+45.67e3` |
| **-**: | negatives Vorzeichen | `-3.4e-5` |
| **+**: | Addition | `1.2 + 9.8 (= 11.0)` |
| **-**: | Subtraktion | `1.2 - 9.8 (= -8.6)` |
| *****: | Multiplikation | `6.1 * 5.0 (= 30.5)` |
| **/**: | Gleitkomma-Division | `9.0 / 2.0 (= 4.5)` |
| **%**: | Gleitkomma-Restbildung | `4.0 % 2.2 (= 1.8)` |

Rundungsfehler möglich (1.79999)!

# Operatoren / boolesche Arithmetik

- Operanden vom Typ `boolean`
- gelieferter Wert vom Typ `boolean`

| | | | |
|---|---|---|---|
| **`!`** | Negation | `!true` | `(= false)` |
| **`& bzw. &&`** | Konjunktion | `true && false` | `(= false)` |
| **`\| bzw. \|\|`** | Disjunktion | `true \|\| false` | `(= true)` |
| **`^`** | Exkl. Oder | `true ^ true` | `(= false)` |

# Operatoren / Vergleiche

- Operanden vom Typ `int`, `short` , `long`, `float`, `double` oder `char`
- gelieferter Wert vom Typ `boolean`

| | | |
|---|---|---|
| **==**: | Gleichheit | `4 == 5` (= `false`) |
| **!=**: | Ungleichheit | `6 != 7` (= `true`) |
| **<**: | Kleiner | `-2. < 0.1` (= `true`) |
| **<=**: | Kleiner-Gleich | `3 <= 3` (= `true`) |
| **>**: | Größer | `'a' > 'b'` (= `false`) |
| **>=**: | Größer-Gleich | `1 >= -1` (= `true`) |

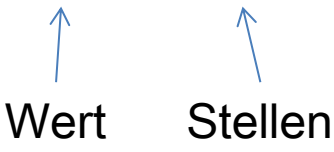
# Operatoren / Bitoperatoren

| | | | |
|---|---|---|---|
| ~ | Negation | `~0000` | `(= 1111)` |
| & | Und | `0101 & 0110` | `(= 0100)` |
| \| | Oder | `0101 \| 0110` | `(= 0111)` |
| ^ | Exkl. Oder | `0101 ^ 0110` | `(= 0011)` |

**>>** **vorzeichenbehaft. Rechtsshift** `8 >> 3 (= 1)`

**>>>** **vorzeichenloser Rechtsshift** `-7 >>> 3`

**<<** **Linksshift** `4 << 3 (= 32)`



# Operatoren / Zuweisung

- Operanden von beliebigem Typ
- gelieferter Wert vom Typ der Operanden

```
<Zuweisung> ::= <Variablenname> "=" <Ausdruck> ";"
```

```
int i = 0, j = -34; // Initialisierung
i = 45;             // Zuweisung
i = i + 3;
j = i = j + i*3;

double pi = 3.1415;
pi = 3.1415631;
```

# Operatoren / Zuweisung / Abkürzungen

| | | |
|---|---|---|
| `i++` | `i = i + 1` | `++i` |
| `i--` | `i = i - 1` | `--i` |
| `i += <expr>` | `i = i + (<expr>)` | |
| `i -= <expr>` | `i = i - (<expr>)` | |
| `i *= <expr>` | `i = i * (<expr>)` | |
| `i /= <expr>` | `i = i / (<expr>)` | |
| `i %= <expr>` | `i = i % (<expr>)` | |

# Operatoren / Bedingungsoperator

- ternärer Operator
- erster Operand vom Typ `boolean`
- Operanden zwei und drei typkonform

```
int i = 45, j = 46;

int k = i<j ? i-1 : j+2;
```

```
int k = 0;
if (i<j)
    k = i-1;
else
    k = j+2;
```

# Operatoren / Typumwandlungen (1)

- einige Typumwandlungen **implizit**:

**short ⟶ int ⟶ long**

```
short s = 3;
int i = s * 4 + 5;
long l = i * 6 * s;
```

**float ⟶ double**

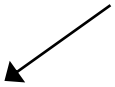
```
float f = 3.4F;
double d = f * 2.3;
```

**Ganzzahl ⟶ Gleitkomma**

```
float g = 3 + i;
```

**char ⟶ int**

Unicode-Dezimalrepräsentation



```
int b = 'a';
int anzahlB = 'z' - 'a' + 1;
```

# Operatoren / Typumwandlungen (2)

- Explizite Typumwandlung (Typecast):

```
<Cast> ::= "(" <Typ> ")"
```

```
double d = 7.99;
int i = (int) d;  // i == 7

int zahl = 33000;
short s = (short)zahl; // Abschneiden der oberen Bits
                       // s == -32536

char ch = (char)('a' + 1); // ch == 'b'
```

- Nicht alle Typumwandlungen sind erlaubt:

```
int wahr = (int)true;  // Fehler!!!
```

# Operatoren / Präzedenz

Liste mit absteigender Präzedenz:

| | |
|---|---|
| ▪ Postfix-Operatoren | `[], ., ++, --` |
| ▪ unäre Operatoren | `+, -, ~, !` |
| ▪ Erzeugung / Typumwandlung | `new, (type)` |
| ▪ Multiplikationsoperatoren | `*, /, %` |
| ▪ Additionsoperatoren | `+, -` |
| ▪ Gleichheitsoperatoren | `==, !=` |
| ▪ logisches Und | `&&` |
| ▪ logisches Oder | `\|\|` |
| ▪ Bedingungsoperator | `?:` |
| ▪ Zuweisungsoperatoren | `=, +=, *=, ...` |

Abänderung der Präzedenz durch Klammernsetzung möglich!

# Operatoren / Assoziativität

- wichtig bei Operatoren gleicher Präzedenz
- alle unären Operatoren sind rechts-assoziativ
- alle binären Operatoren außer den Zuweisungsoperatoren sind links-assoziativ
- Zuweisungsoperatoren sind rechts-assoziativ
- Assoziativität ist bspw. von Bedeutung bei Funktionen mit Seiteneffekten!

```
double d1 = 20.0 / 8.0 / 2.0;   // d1 == 1.25
double d2 = 20.0 / (8.0 / 2.0); // d2 == 5.0
```

```
i = j = k / z * 5   ⟷   i = (j = ((k / z) * 5))
```

# Operatoren / Auswertung

- jeder Operand wird vor der Ausführung des Operators ausgewertet;
  Ausnahmen: `&&`, `||` und `?` („verzögerte Auswertung“):

```
p && q    ⟷   p ? q : false

p || q    ⟷   p ? true : q

o1 ? o2 : o3   werte o1 aus; falls true, werte
               o2 ansonsten o3 aus
```

- wichtig bei Seiteneffekten!

# Zeichenketten

- **`String`**: Datentyp für Zeichenketten
- Referenzdatentyp (Strings sind Objekte); kein Standarddatentyp
- Initialisierung explizit mit `""` (Defaultwert ist `null`)

```
String h = "hello";
String w = "world";
String hw = h + " " + w + "!";
  // + ist Konkatenation-Operator: "hello world!"

int zahl = 4711;
String duft = "koelnisch Wasser = " + zahl;
  // Typumwandlung: "koelnisch Wasser = 4711";

String s1 = "hello" + zahl;
String s2 = "hello" + zahl;
boolean vgl1 = s1 == s2; // liefert false (Ref.vergl.)
boolean vgl2 = s1.equals(s2); // vergleicht Zeichenketten
```

# Eingabe mit Klasse IO

- `readShort` → `short`
- `readInt` → `int`
- `readLong` → `long`
- `readFloat` → `float`
- `readDouble` → `double`
- `readChar` → `char`
- `readBoolean` → `boolean`
- `readString` → `String`

Beispiel:

```
float zahl = IO.readFloat("Reelle Zahl: "); // . statt , !!!
String zeichenkette = IO.readString("String: ");
```

# Zusammenfassung

- Datentypen: Zusammenfassung von Wertebereichen und Operationen
- Variablen: logische Behälter zur Speicherung von Werten
- Ausdruck: Verarbeitungsvorschrift, deren Ausführung einen Wert liefert
- Operatoren: Verknüpfung von Ausdrücken