

Datentypen und Variablen

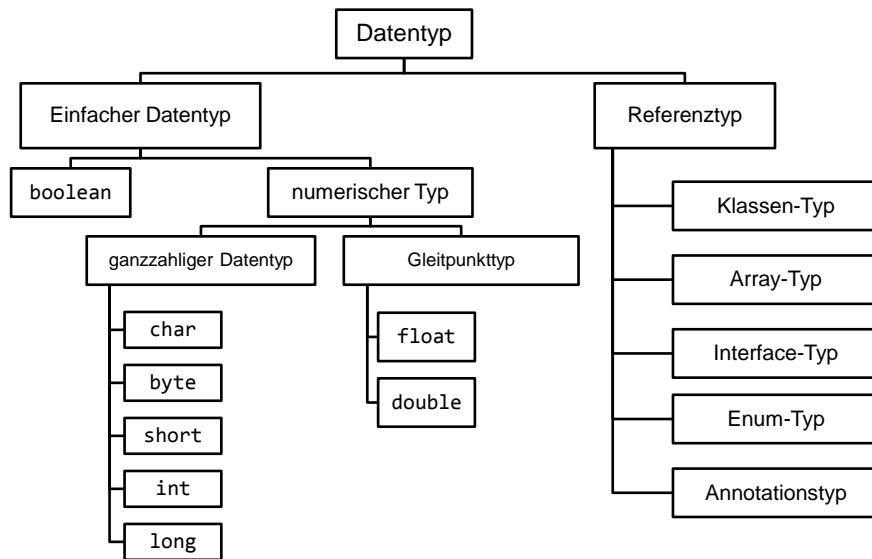
Prof. Dr. Helmut G. Folz



Datentypen

- Datentypen legen fest, wie Variablen intern gespeichert werden und wie sie sich verhalten.
- Java kennt nur zwei Arten von Datentypen:
 - ⇒ **einfache (elementare) Datentypen** und
 - ⇒ **Referenzdatentypen**.
- Ein einfacher Datentyp repräsentiert einen einzigen Wert, entweder eine Zahl, ein Zeichen oder einen Wahrheitswert.
- Referenztypen können sein **Klassen**, **Interfaces**, **Arrays**, **Enums** oder **Annotationen**.
- Während einfache Datentypen vordefiniert sind, sind Referenztypen benutzerdefinierte Typen (später mehr)

Klassifikation der Datentypen



Variablen

- Datentypen legen fest, wie Variablen intern gespeichert werden und wie sie sich verhalten.
- Alle Variablen eines Datentyps haben die selbe Darstellung im Arbeitsspeicher, d. h. die selbe Anzahl an Bytes und die selbe Interpretation der einzelnen Bits.

Typen von Variablen sind

- **Attribute**, die innerhalb der Klasse definiert sind und entweder mit der Klasse oder mit einem Objekt erzeugt werden.
- **Lokale Variablen**, die innerhalb einer Methode oder innerhalb eines Anweisungsblocks definiert sind

Überblick über einfache Datentypen

Typ	repräsentiert	Wertebereich
boolean	true oder false	true und false
char	16-Bit-Unicode-Zeichen	0 bis 65535 (Unicode-Zeichen)
byte	8 Bit-Ganzzahl	-128 127
short	16-Bit-Ganzzahl	-32768 32767
int	32-Bit-Ganzzahl	-2147483648 2147483647
long	64-Bit-Ganzzahl	-9223372036854775808 9223372036854775807
float	32 Bit-Gleitpunktzahl	+/- 3.40282347E+38 +/- 1.40239846E-45
double	64 Bit-Gleitpunktzahl	+/- 1.79769313486231570E+308 +/- 4.940656458412465544E-324

Die ganzzahligen Datentypen byte, short, int und long

- Bei Java werden die Datentypen **byte**, **short**, **int** und **long** auf allen Plattformen intern gleich dargestellt. Sie speichern ganze Zahlen in der sogenannten Zweierkomplementdarstellung und umfassen 8, 16, 32 bzw. 64 Bits.
- Beispiel: Gegeben sei folgendes Bitmuster

1	0	1	0	0	1	1	1
-2 ⁷	+2 ⁶	+2 ⁵	+2 ⁴	+2 ³	+2 ²	+2 ¹	+2 ⁰

- Die Zahl errechnet sich zu
$$-1 \cdot 128 + 0 \cdot 64 + 1 \cdot 32 + 0 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = -89$$
- Das höchste Bit der Zweierkomplementzahl gibt das Vorzeichen an. Ist es 0, so ist die Zahl positiv, ist es 1, so ist die Zahl negativ.
- Das Negative einer Zahl wird gebildet, indem alle Bits konvertiert werden und dann auf das Ergebnis 1 addiert wird.

Beispiel: Limits1

```
/** Limits1: Maximale und Minimale Werte der einfachen Typen
 */
public class Limits1 {
    public void start() {
        System.out.println("byte : " + Byte.MIN_VALUE
                           + "... " + Byte.MAX_VALUE);
        System.out.println("short: " + Short.MIN_VALUE
                           + "... " + Short.MAX_VALUE);
        System.out.println("int : " + Integer.MIN_VALUE
                           + "... " + Integer.MAX_VALUE);
        System.out.println("Long : " + Long.MIN_VALUE
                           + "... " + Long.MAX_VALUE);
    }
    public static void main(String[] args) {
        new Limits1().start();
    }
}
```

```
/* Ausgabe
byte : -128...127
short: -32768...32767
int : -2147483648...2147483647
long : -9223372036854775808...9223372036854775807
```

Prof. Dr. H. G. Folz

Ganzzahlige Literale

- Literale sind konstante Ausdrücke, die einen festen Wert repräsentieren. Jedes Literal hat einen eindeutigen zugehörigen Datentyp
- Literale vom Typ int:

12	-12	0	1234	Dezimaldarstellung
033				Oktaldarstellung
0x1b				Hexadezimaldarstellung
0b10101010				Binärdarstellung
- Literale vom Typ long:

123L	1234567L	-45678L	Dezimaldarstellung
07777777L			Oktaldarstellung
0xffffffffL			Hexadezimaldarstellung
0b10101010L			Binärdarstellung

Erweiterungen seit Java 7

Binäre Literale sind möglich

```
// An 8-bit 'byte' value:
byte aByte = (byte)0b00100001;

// A 16-bit 'short' value:
short aShort = (short)0b1010000101000101;

// Some 32-bit 'int' values:
int anInt1 = 0b10100001010001011010000101000101;
int anInt2 = 0b101;
int anInt3 = 0B101; // The B can be upper or lower case.

// A 64-bit 'long' value. Note the "L" suffix:
long aLong =
0b1010000101000101101000010100010110100001010001011010000101000101L;
```

Erweiterungen seit Java 7

Unterstriche sind zwischen Ziffern erlaubt

```
long creditCardNumber = 1234_5678_9012_3456L;

long socialSecurityNumber = 999_99_9999L;

long hexBytes = 0xFF_EC_DE_5E;

long hexWords = 0xCAFE_BABE;

long maxLong = 0x7fff_ffff_ffff_ffffL;

byte nybbles = 0b0010_0101;

long bytes = 0b11010010_01101001_10010100_10010010;
```

Die Gleitpunkt-Datentypen float und double

- Mit Gleitpunktzahlen versucht man die reellen Zahlen der Mathematik darzustellen. Bei Java werden auf allen Rechnerplattformen die Gleitpunkt-Datentypen nach dem Standard IEEE 754 dargestellt.

Anzahl der Bits		
	float	double
Vorzeichen	1	1
Exponent	8	11
Mantisse	23	52

Die Gleitpunkt-Datentypen float und double

Interne Darstellung bei float:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
V	Exponent E								Mantisse M																						

- V: Vorzeichenbit (0 : positiv, 1 : negativ)
- Exponent E zwischen -127 und +127
- Mantisse $M = m_1 m_2 \dots m_k$
- Darstellung der Zahl:
 $\pm 1, m_1 m_2 \dots m_k \cdot 2^E$ bzw.

$$(-1)^V \cdot (1 + m_1 2^{-1} + m_2 2^{-2} + \dots + m_k 2^{-k}) \cdot 2^E$$

- E ergibt sich aus den Bits 30-23 durch Subtraktion von 127

Die Gleitpunkt-Datentypen float und double

V	Exponent	Mantisse	Wert
0	00000000	00000000 00000000 00000000 00000000	0.0
0	01111111	00000000 00000000 00000000 00000000	1.0
0	10000000	00000000 00000000 00000000 00000000	2.0
0	01111110	00000000 00000000 00000000 00000000	0.5
0	01111111	10000000 00000000 00000000 00000000	1.5
1	01111111	11000000 00000000 00000000 00000000	-1.75
0	01111011	10011001 10011001 10011001 10011001	0.1
0	11111111	*	INFINITY
0	11111111	*1*	NaN

$$-1.75 = (-1)(1 + \frac{1}{2^1} + \frac{1}{2^2} + 0 \dots)2^0$$

$$0.1 = (1 + 1 \cdot \frac{1}{2^1} + 0 \cdot \frac{1}{2^2} + 0 \cdot \frac{1}{2^3} + 1 \cdot \frac{1}{2^4} + 1 \cdot \frac{1}{2^5} + 0 \cdot \frac{1}{2^6} \dots)2^{-4}$$

Beispiel: Limits2 (1)

```
/**
 * Limits2: Maximale und Minimale Werte der einfachen Typen
 */
public class Limits2 {
    public void start() {
        System.out.println("float : " + Float.MIN_VALUE
                           + "..." + Float.MAX_VALUE);
        System.out.println("double: " + Double.MIN_VALUE
                           + "..." + Double.MAX_VALUE);

        System.out.println("Werte fuer Unendlich"
                           + " und Not a Number");
        System.out.println("float : " + Float.NEGATIVE_INFINITY
                           + "..." + Float.POSITIVE_INFINITY);
        System.out.println("double: " + Double.NEGATIVE_INFINITY
                           + "..." + Double.POSITIVE_INFINITY);
        System.out.println("float : " + Float.NaN);
        System.out.println("double: " + Double.NaN);
    }
}
```

Beispiel: Limits2 (2)

```
System.out.println("1.0 / 0.0 = " + 1.0 / 0.0);
System.out.println("-1.0 / 0.0 = " + -1.0 / 0.0);
System.out.println("0.0 / 0.0 = " + 0.0 / 0.0);
System.out.println("1.0/0.0 + -1.0/0.0 = "
    + (1.0/0.0 + -1.0/0.0));
}
public static void main(String[] args) {
    new Limits2().start();
}
}
```

float : 1.4E-45...3.4028235E38
double: 4.9E-324...1.7976931348623157E308
Werte fuer Unendlich und Not a Number
float : -Infinity...Infinity
double: -Infinity...Infinity
float : NaN
double: NaN
1.0 / 0.0 = Infinity
-1.0 / 0.0 = -Infinity
0.0 / 0.0 = NaN
1.0/0.0 + -1.0/0.0 = NaN

Prof. Dr.

Gleitpunktliterale

- Gleitpunkt-Literale werden in Java wie folgt beschrieben:
 - Mantisse bestehend aus:
 - Vorzeichen (optional)
 - Vorkommastellen
 - Dezimalpunkt
 - Nachkommastellen
 - Exponential-Anteil bestehend aus:
 - e oder E und
 - ganzzahliger Exponent (optional)
 - angehängte Typkennung zur Unterscheidung von float und double
 - Suffix f, F, d, D
 - Literale ohne Suffix sind automatisch vom Typ double
- Beispiele:

-123.789	1.2E6	.5d	-1.123e-5D	double
1.23f	-123.45e12F			float

Gleitpunktliterale

Unterstriche zwischen Ziffern sind erlaubt

```
float pi = 3.14_15F;  
double x = 1.234_567_890;
```

Beispiel: DoubleTest1 (1)

```
/** DoubleTest1: Ungenauigkeit der Gleitpunkt-Arithmetik  
 */  
public class DoubleTest1 {  
    public void start() {  
        double x = -1.0;  
        while (x <= 0.0) {  
            System.out.println(x);  
            x = x + 0.1;  
        }  
        System.out.println();  
  
        System.out.println("0.01 - 0.1 * 0.1 = "  
                           + (0.01 - 0.1 * 0.1));  
    }  
  
    public static void main(String[] args) {  
        new DoubleTest1().start();  
    }  
}
```

Beispiel: DoubleTest1 (2)

```
-1.0  
-0.9  
-0.8  
-0.7000000000000001  
-0.6000000000000001  
-0.5000000000000001  
-0.40000000000000013  
-0.30000000000000016  
-0.20000000000000015  
-0.10000000000000014  
-1.3877787807814457E-16
```

```
0.01 - 0.1 * 0.1 = -1.734723475976807E-18
```

Beispiel: DoubleTest2

```
public class DoubleTest2 {  
    public void start() {  
        double a = 1.0f,  
               b = 1.0E16,  
               c = -b;  
        double x = (a + b) + c;  
        double y = a + (b + c);  
        System.out.println("(a + b) + c = " + x);  
        System.out.println("a + (b + c) = " + y);  
    }  
  
    public static void main(String[] args) {  
        new DoubleTest2().start();  
    }  
}  
/* Ausgabe:  
(a + b) + c = 0.0  
a + (b + c) = 1.0  
*/
```

Der logische Datentyp boolean

- Der Datentyp **boolean** kann nur die beiden Wahrheitswerte **true** oder **false** annehmen.
- **true** und **false** sind die möglichen Konstanten dieses Datentyps und keine Schlüsselwörter

Beispiel: BooleanTest

```
public class BooleanTest {
    public void start() {
        int i = 2, j = 3;
        boolean b1 = i > j;
        boolean b2 = i < j;

        System.out.println("b1 = " + b1);
        System.out.println("b2 = " + b2);
    }

    public static void main(String[] args) {
        new BooleanTest().start();
    }
}

/* Ausgabe:
b1 = false
b2 = true
*/
```

Vergleichsoperatoren

Das Ergebnis eines Vergleichs ist immer ein Ausdruck vom Typ **boolean**

Operator	Bedeutung
ausdruck1 == ausdruck2	Gleichheit zweier Ausdrücke prüfen
ausdruck1 != ausdruck2	Ungleichheit zweier Ausdrücke prüfen
ausdruck1 < ausdruck2	Ist ausdruck1 kleiner als ausdruck2 ?
ausdruck1 > ausdruck2	Ist ausdruck1 größer als ausdruck2 ?
ausdruck1 <= ausdruck2	Ist ausdruck1 kleiner oder gleich ausdruck2 ?
ausdruck1 >= ausdruck2	Ist ausdruck1 größer oder gleich ausdruck2 ?

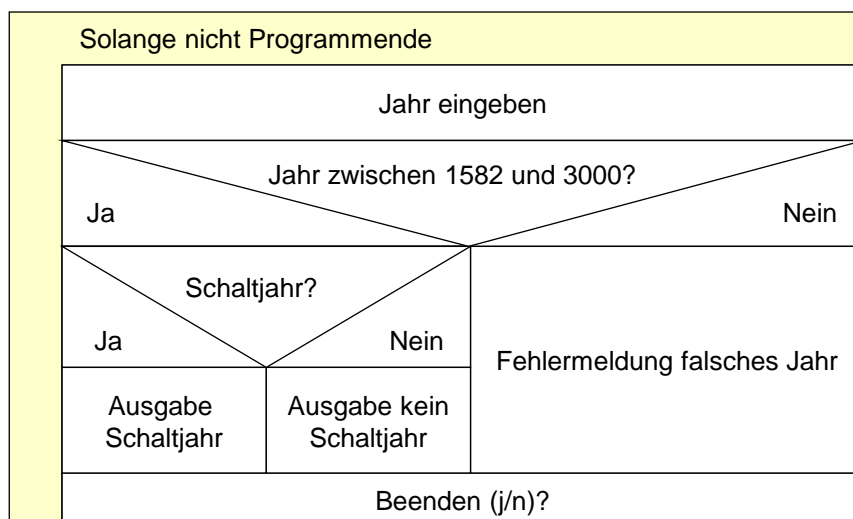
Logische Operatoren

Operator	Bedeutung
ausdruck1 && ausdruck2	logisches Und angewandt auf zwei boolean-Ausdrücke
ausdruck1 ausdruck2	logisches Oder angewandt auf zwei boolean-Ausdrücke
!ausdruck1	Verneinung des boolean-Ausdruckes ausdruck1

Klasse: SchaltjahrTest

- Es soll eine Klasse zur Eingabe von Jahren geschrieben werden.
- Dabei soll jeweils überprüft werden ob ein Jahr in einem vorgegebenen Intervall liegt und ob es ein Schaltjahr ist.
- Prüfungen:
 - ⇒ Jahr liegt zwischen 1582 und 3000
 - ⇒ Entscheiden ob Schaltjahr oder nicht

Klasse: SchaltjahrTest



Beispiel: SchaltjahrTest (1)

```
/**
 * SchaltjahrTest: Testen der Schaltjahrbedingung bei einer
 * Jahreszahl
 */
public class SchaltjahrTest {
    private Scanner input = new Scanner(System.in);

    public void start() {
        boolean programmEnde = false;
        char antwort;
        int jahr;

        while (!programmEnde) {
            System.out.println("Jahr zwischen 1582 und 3000" +
                               " eingeben: ");
            jahr = input.nextInt();
        }
    }
}
```

Beispiel: SchaltjahrTest (2)

```
        if (jahr < 1582 || jahr > 3000) {
            System.out.println(jahr
                               + " liegt nicht zwischen 1582 und 3000!");
        } else {
            if ((jahr % 4 == 0 && jahr % 100 != 0)
                || jahr % 400 == 0)
                System.out.println(jahr +
                                   " ist ein Schaltjahr!");
            else
                System.out.println(jahr +
                                   " ist kein Schaltjahr!");
        }

        System.out.print("Beenden (j/n) ");
        antwort = input.next().charAt(0);
        programmEnde = antwort == 'j';
    }
}
```

Der Datentyp char

- Für die Darstellung von Zeichen aus dem UNICODE-Zeichensatz wird der Datentyp char verwendet.
- Eine char-Variable belegt intern 2 Bytes. Zeichen werden intern als Binärzahlen zwischen 0 und 65535 gespeichert.

Zeichen-Literale (Zeichenkonstanten)

- Ein Zeichenliteral ist ein Zeichen eingeschlossen in einfache Anführungszeichen. Es gibt verschiedene Möglichkeiten, Zeichenliterale darzustellen.

- Als Zeichen:

```
char c = 'x';           // Der Kleinbuchstabe x
char leer = ' ';        // Das Leerzeichen (Blank)
```

- Als Zahl:

```
char eins = 49;        // ASCII-Code für das Zeichen '1'
```

Beispiel: CharTest1

```
char eins = 49;
char zwei = (char)(eins + 1);

// Oktaldarstellungen
char einsOkt  = '\61'; // das Zeichen '1' oktal
char grossesA = '\101'; // das Zeichen 'A'
char kleinesA = '\141'; // das Zeichen 'a'
char escape   = '\33'; // das escape-Zeichen
char lf       = '\12'; // Zeilenendezeichen (Linefeed)
char cr       = '\15'; // Wagenrücklauf (Carriage Return)

// Unicode-Ersatzdarstellungen
char einsHex  = '\u0031'; // das Zeichen '1' hexadezimal
char grossesAX = '\u0041'; // das Zeichen 'A'
char kleinesAX = '\u0061'; // das Zeichen a
char escapeX   = '\u001b'; // das escape-Zeichen
char aleph     = '\u05D0'; // Unicode-Literal
```

Ersatzdarstellungen

Zeichen	Bedeutung	ASCII-Name	Unicode
\b	Backspace	BS	\u0008
\f	Seitenvorschub (Formfeed)	FF	\u000c
\n	neue Zeile (Linefeed)	LF	\u000a
\r	Zeilenrücklauf (Carriage Return)	CR	\u000d
\t	Tabulator	HT	\u0009
\\	Backslash		\u005c
\'	das Zeichen '		\u0027
\"	das Zeichen "		\u0022
\ooo	Oktalcode, von \0 bis \377		\u0000 - \u00ff

Beispiel: CharTest2 (1)

```
public class CharTest2 {
    public void start() {
        int i = 0;
        char c = ' ';

        while ( c <= '~' ) {
            System.out.print(c);
            c = (char)(c + 1);
            i = i + 1;
            if ( i == 64 ) {
                System.out.println();
                i = 0;
            }
        }
        System.out.println();
    }

    public static void main(String[] args) {
        new CharTest2().start();
    }
}
```


Beispiel: CharTest2 (2)

```
/* Ausgabe
!"#$%&'()*+,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~
*/
```

Erlaubte Operatoren für Zeichen

=	Zuweisung
== != < > <= >=	Vergleichsoperatoren verglichen wird jeweils der interne Binärkode

Zeichenketten-Literale

- Zeichenketten-Literale sind in Java Objekte der Standardklasse **String** (dazu später mehr).

"Hallo Welt! \n"

ist also ein Objekt und intern anders dargestellt als dies in anderen Programmiersprachen (z. B. C/C++) üblich ist.

Typkonvertierungen

- Welche der folgenden Zuweisungen sind erlaubt?

```
int i = 100;  
long l = 1234567890L;  
float f = 1.234E12f;  
double d = 3.1E123;
```

```
i = l;  
l = i;  
f = d;  
d = f;  
i = f;
```

Typkonvertierungen

- Java ist wesentlich strenger typisiert als C++. Automatische Konvertierungen zwischen verschiedenen einfachen Datentypen sind nur bei Typerweiterungen möglich:
byte → short → int → long → float → double
- Zum expliziten Konvertieren gibt es nur den sogenannten C-Cast-Operator:

Syntax: *(zieltyp)ausdruck*

```
double pi = 3.1415926;  
int i = (int) pi;
```

Probleme bei Typkonvertierungen

```
public class TypTest {  
    public void start() {  
        int i = 100;  
        long l = 1_234_567_890_123L;  
        float f = 1.234E12f;  
        double d = 3.1E123;  
  
        i = (int)l; // i = 1912276171  
        l = i;      // l = 1912276171  
        f = l;      // f = 1.91227622E9  
        l = (long)f; // l = 1912276224  
        f = (float)d; // f = infinity  
        d = f;      // d = infinity  
        i = (int)f;  // i = 2147483647  
        d = 1.5e100; // d = 1.5E100  
        i = (int)d;  // i = 2147483647  
  
        ...  
    }  
}
```