Vorlesung an der

Berufsakademie Oldenburg

Unterrichtseinheit 13

Typen, Operatoren und Ausdrücke

Dr. Dietrich Boles

Gliederung

- Datentypen
- Variablen
- Ausdrücke
- Operatoren
- Strings
- Eingabeanweisungen
- Beispiele

- Aufgaben

Datentypen (1)

- Daten haben unterschiedliche Eigenschaften:
  - Ganze Zahlen: 2, 4711, -45, ...
  - Reelle Zahlen: -3.4, -56.789678, -3.4e-10, 45.567e123, ...
  - Buchstaben: 'a', 'b', ...
  - Zeichenketten: "hello world!", ...
  - ...
  - Unsinn: `3 + "hello"`

- Unterschiedliche Daten benötigen unterschiedlich viel Platz
  - boolesche Werte: 1 Bit
  - ASCII-Zeichen: 7 Bit
  - ...

- Motivation für Datentypen:
  - Überprüfung "sinnvoller" Operationen
  - Reservierung von "passendem" Speicherplatz

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)

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

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

Datentypen (3)

	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 (1)

- Initialisierung von Variablen
  - implizit durch Default-Werte
  - explizit mit Hilfe von Literalen bzw. Ausdrücken
  - Empfehlung: Variablen immer explizit initialisieren!

- Literale: Typ-spezifische Konstanten

  - `boolean`:
    `true`, **`false`**

  - `int`:
    Zeichenketten aus
    - dezimalen **`29`**
    - oktalen (führende 0): **`035`**
    - hexadezimalen Ziffern (führendes 0x): **`0x1D`**

Variablen / Initialisierung (2)

- `long`:

  `int`-Literal mit nachgestelltem `l` oder `L`: **`29L  4321l`**

- `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' '\"' '\\'`**

- *`Zeichenketten`*:

  Zeichen zwischen zwei Anführungszeichen: **`"hello\n world"`**

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

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

Operatoren / Integer-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)`

Rundungsfehler möglich!

Operatoren / boolesche Arithmetik

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

&&:	Konjunktion	`true && false (= false)`
||:	Disjunktion	`true || false (= true)`
!:	Negation	`!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 / 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 / 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`
  - `float` → `double`
  - arithmetisch → Gleitkomma
  - `char` → `int`

```
short s = 3;
int i = s;
long l = i = s;

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

float g = 3 + i;

int b = 'a';
'z' - 'a'
```

← ASCII-Repräsentation

Operatoren / Typumwandlungen (2)

- Explizite Typumwandlung (Typcast):

```
<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
```

- 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
- der ternäre Bedingungsoperator (`?:`) ist rechts-assoziativ
- Assoziativität ist bspw. von Bedeutung bei Funktionen mit Seiteneffekten!

```
i = j *= k + 78;   ⟷   i = (j *= (k + 78));
i = j * k / 5;     ⟷   i = (j * k) / 5;
```

Operatoren / Auswertungsreihenfolge

- Operanden werden immer von links nach rechts ausgewertet
- wichtig bei Seiteneffekten!
- jeder Operand wird vor der Ausführung des Operators ausgewertet;
  Ausnahmen: `&&`, `||` und `?:`

```
p && q    ⟷    p ? q : false

p || q    ⟷    p ? true : q

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

Strings

- **`String`**: Datentyp für Zeichenketten
- Referenzdatentyp (Strings sind Objekte); kein Standarddatentyp

```
String h = "hello";
String w = "world";
String h_w = h + " " + w + "!";
  // + ist Concatenation-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
```

Strings / Beispielprogramm

- Aufgabe:

  Berechnung des Quadrats einer eingegebenen Zahl

- Programm:

```
class Quadratzahl {
  public static void main(String[] args) {

    int zahl = IO.readInt("Zahl: ");
    int quadrat = zahl * zahl;
    String ausgabe = zahl + " * " + zahl + " = " + quadrat;
    System.out.println(ausgabe);

  }
}
```

Eingabeanweisungen

```
class Eingaben {
  public static void main(String[] args) {

    // pro Typ zwei Eingabefunktionen

    int zahl = IO.readInt("Zahl: ");
    char c = IO.readChar("Buchstabe: ");
    double pi = IO.readDouble("PI: ");
    long gross = IO.readLong();
    String antwort = IO.readString("alles klar?");
    ...

  }
}
```

Beispiel 1

```
class Kegelvolumen {
  public static void main(String[] args) {

    System.out.println(
      "Berechnung des Volumens eines Kegels");
    double pi = 3.1415;
    double hoehe = IO.readDouble("Hoehe: ");
    double radius = IO.readDouble("Radius: ");
    System.out.println("Volumen = " +
      1.0/3.0 * pi * radius * radius * hoehe);

  }
}
```

Beispiel 2

```
class ZinsesZins {
  public static void main(String[] args) {

    System.out.println("Berechnung des ZinsesZins");
    double kapital = IO.readDouble("Kapital: ");
    double zinssatz = IO.readDouble("Zinssatz: ");
    double neuKapital =
      kapital * (1.0 + zinssatz / 100.0) *
                (1.0 + zinssatz / 100.0) *
                (1.0 + zinssatz / 100.0) *
                (1.0 + zinssatz / 100.0);
    System.out.println(
      "Kapital in 4 Jahren = " + neuKapital);

  }
}
```

Beispiel 3

Schreiben Sie ein Programm "*Passt*", das zunächst einen Kleinbuchstaben und dann einen Großbuchstaben einliest und dann überprüft, ob diese zueinander passen, d.h. Kleinbuchstabe entspricht Großbuchstaben.

Beispiel:

```
$ java Passt
Klein:
h<CR>
Gross:
H<CR>
passen
$ java Passt
Klein:
d<CR>
Gross:
E<CR>
passen nicht
```

```
class Passt {
  public static void main(String[] a) {
    char klein = IO.readChar("Klein:");
    char gross = IO.readChar("Gross:");
    if (klein-'a' == gross-'A')
      IO.println("passen");
    else
      IO.println("passen nicht");
  }
}
```

Aufgabe 1

Schreiben Sie ein Java-Programm `NachKommaStellen`, das die Eingabe eines `double`-Wertes erwartet und die ersten 4 Nachkommastellen als `int`-Wert auf den Bildschirm ausgibt.

Beispiele:

```
$ java NachKommaStellen
Eingabe: 12.345678
Ausgabe: 3456

$ java NachKommaStellen
Eingabe: -2.000134
Ausgabe: 1
```

Aufgabe 2

Schreiben Sie ein Java-Programm `Morse`, das die Eingabe eines `char`-Wertes über die Tastatur erwartet. Das Programm soll anschließend die Eingabe überprüfen und falls es sich um eine Ziffer handelt, diese ins Morse-Alphabet übersetzen und die entsprechende Morse-Kodierung auf den Bildschirm ausgeben. Hier das entsprechende Morse-Alphabet:

```
1 .----   2 ..---  3 ...--  4 ....-  5 .....
6 -....   7 --...  8 ---..  9 ----.  0 -----
```

Beispiele:

```
$ java Morse
Eingabe: 4
Ausgabe: ....-

$ java Morse
Eingabe: a
Ausgabe: ungueltiges Zeichen
```