

Einfache Rechenstrukturen und Kontrollfluß

Martin Wirsing

in Zusammenarbeit mit
Michael Barth, Philipp Meier und Gefei Zhang

11/04

Ziele

- Verstehen der Grunddatentypen von Java
- Verstehen von Typkonversion in Java
- Lernen lokale Variablen und Konstanten zu initialisieren
- Verstehen der Speicherorganisation von lokalen Variablen
- Lernen imperative Programme in Java mit
 Zuweisung, Block, Fallunterscheidung, Iteration
zu schreiben

Gleitpunktzahlen

- float 4 Byte bis ca. 10^{32}
 - double 8 Byte bis ca. 10^{308}
- nach IEEE-754--Standard (1985)

Beispiele:

- double: 6.22, 622E-2 , 62.2e-1
- float: 6.22F, 622E-2F, 62.2e-1f

Arithmet. Operationen und Vergleichsoperationen

* Multiplikation, / Division, % Modulo (Rest)

+ Addition, - Subtraktion

> größer, >= größer oder gleich

< kleiner, <= kleiner oder gleich

== gleich, != nicht gleich

(= **Zuweisung** wird als Gleichheit geschrieben)

Typkonversion

„Kleiner-Beziehung“ zwischen Datentypen:

`byte < short < int < long < float < double`

Java konvertiert Ausdrücke automatisch in den allgemeineren Typ.

Beispiele:

<code>1 + 1.7</code>	ist vom Typ <code>double</code>
<code>1.0f + 1</code>	ist vom Typ <code>float</code>
<code>1.0f + 1.0</code>	ist vom Typ <code>double</code>

Typkonversion

Type Casting:

Erzwingen der Typkonversion (zum spezielleren Typ `type`) durch
Vorstellen von „ `(type)` “

Beispiele:

<code>(byte) 3</code>	ist vom Typ <code>byte</code>
<code>(int) (2.0 + 5.0)</code>	ist vom Typ <code>int</code>
<code>(float) 1.3e-7</code>	ist vom Typ <code>float</code>

Bei der Typkonversion kann Information verloren gehen.

Beispiel:

<code>(int) 5.2</code>	<code>==</code>	<code>5</code>
<code>(int) -5.2</code>	<code>==</code>	<code>-5</code>

Zeichen

- Typ **char** (für character)
- bezeichnet Menge der Zeichen aus dem Unicode-Zeichensatz
- **char** umfaßt ASCII-Zeichensatz mit kleinen und großen

Buchstaben, Zahlen und verschiedenen Sonderzeichen

- Darstellung von Zeichen durch Umrahmung mit Apostroph

Beispiel: `'a'` , `'A'` , `'1'` , `'9'`

- Zeichenketten: werden mit Doppelapostroph umrahmt und sind

vom Typ `String` (eine Klasse): „Wirsing“, „Info“

M. Wirsing: Einfache Rechenstrukturen und Kontrollfluß

Boole'sche Werte

Der Typ `boolean` hat genau zwei Werte, `true` und `false`.

Boole'sche Operatoren

!	strikte Negation
&	strikte Konjunktion („und“, auch bitweise Addition)
^	strikte Disjunktion („entweder-oder“)
	strikte Adjunktion („oder“)

op strikt bedeutet, dass der Wert von $x \text{ op } y$ undefiniert ist, falls der Wert von x **oder** der Wert von y undefiniert ist.

op sequenziell bedeutet, dass $x \text{ op } y$ von links nach rechts ausgewertet wird und die Undefiniertheit von y keine Rolle spielt, wenn der Wert von $x \text{ op } y$ schon „klar“ ist. Bsp. `false && undef = false`.

&&	sequenzielle Konjunktion
	sequenzielle Adjunktion

M. Wirsing: Einfache Rechenstrukturen und Kontrollfluß

Boole'sche Werte

„entweder-oder“

^	true	false
true	false	true
false	true	false

und

„oder“

,	true	false
true	true	true
false	true	false

Boole'sche Werte

Beispiel für die strikte/sequentielle Konjunktion

```
int teiler = 0;
(teiler != 0) && (100/teiler > 1) == false // Ok
(teiler != 0) & (100/teiler > 1) == false
// Laufzeitfehler
```

Beispiel für die strikte/sequentielle Adjunktion

```
true || (1/0 == 1) == true; // Ok
true | (1/0 == 1) // Laufzeitfehler
```

Deklaration lokaler Variablen

Eine einfache **Deklaration lokaler Variablen** hat die Form

```
<Type> <VarName> = <Expression>;
```

//Deklaration mit Initialisierung

Beispiel:

```
int total = -5;           //total hat den Initialwert -5
```

```
int quadrat = total * total;
```

```
boolean aussage = false;
```

Bemerkung:

Auf die Initialisierung kann verzichtet werden, wenn zur Übersetzungszeit nachgewiesen werden kann, dass die Variable initialisiert wird bevor sie benutzt wird.

Zustand

- Ein Zustand ist eine **Belegung der Variablen mit Werten**.
- Der Zustand der lokalen Variablen wird beschrieben als **Liste von Variablennamen und zugehörigen Werten**.
- Lokale Variablen werden im „**Keller**“ (engl. „**Stack**“) gespeichert.

Beispiel:

Textuell: [(total, -5), (quadrat, 25), (aussage, false)]

Im Speicher:

total	1000	-5
quadrat	1001	25
aussage	1002	false
} }	} }	}
Lok. Variable	Adresse	Wert

Iterierte Deklaration lokaler Variablen

Beispiel:

```
int total = 17, max = 100, i, j;
```

ist eine Abkürzung für

```
int total = 17;
int max = 100;
int i;
int j;
```

M. Wirsing: Einfache Rechenstrukturen und Kontrollfluß

Deklaration lokaler Konstanten

- Eine Konstante wird durch Angabe des „Modifiers“ **final** deklariert.
Beispiel: `final int TOTAL = 100;`
- Konstanten werden i.a. mit Großbuchstaben geschrieben
- Konstanten sollten (wie auch Variablen) „sprechende“ Namen besitzen
- **Nie „Magic Numbers“ verwenden**

Beispiele:

- Anstelle von 365 im Programm für „Anzahl der Tage im Jahr“ verwende man besser `final int TAGE_PRO_JAHR = 365;`
- Für die mathematischen Größen π und e verwende man anstelle von 3.14159 und 2.7182 besser `Math.Pi` bzw. `Math.E`

M. Wirsing: Einfache Rechenstrukturen und Kontrollfluß

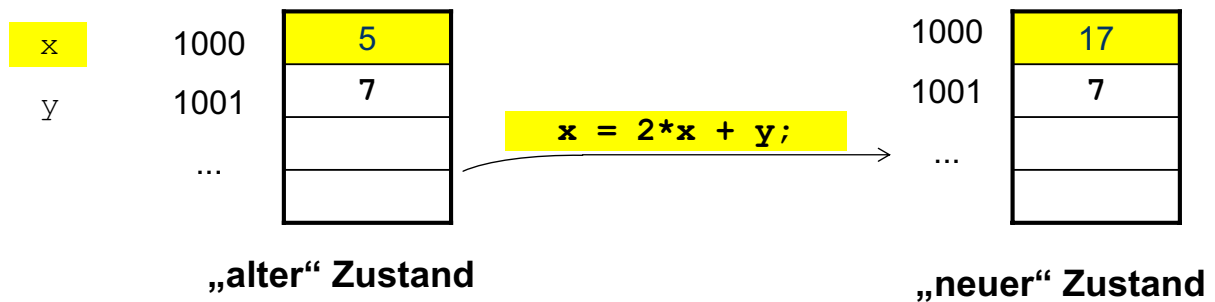
Zuweisung

Bei der Zuweisung

$$\langle \text{VarName} \rangle = \langle \text{Expression} \rangle;$$

wird der Wert w der $\langle \text{Expression} \rangle$ im „alten“ Zustand berechnet und im Nachfolgezustand der Variablen $\langle \text{VarName} \rangle$ als neuer Wert zugewiesen.

Beispiel:



Zuweisung: Textuelle Darstellung

Beispiel textuell:

„alter“ Zustand $s = [(x, 5), (y, 7), (b, true)]$

Zuweisung $x = 2 * x + y;$

„neuer“ Zustand $s = [(x, 17), (y, 7), (b, true)]$

Zuweisung: Abkürzende Schreibweisen

Abkürzungen

<code>x++;</code>	steht für <code>x = x + 1;</code>
<code>x--;</code>	steht für <code>x = x - 1;</code>
<code>x op= <Ausdruck>;</code>	steht für <code>x = x op <Ausdruck></code>

Beispiele

<code>x += y;</code>	steht für <code>x = x + y;</code>
<code>x &&= c;</code>	steht für <code>b = b && c;</code>
<code>x += 3*y;</code>	steht für <code>x = x + 3*y;</code>

Sequenzielle Komposition

Sequenzielle Komposition

wird durch Hintereinanderschreiben ausgedrückt.

Beispiel:

```
int total = 100;
total = total + 100;
```

Bemerkung

Wie bei BNF gibt es bei Java **keinen expliziten** Kompositionsoperator.

Block

- Ein **Block**

```
{ <Statement>
  }
```

fügt mehrere Anweisungen durch geschweifte Klammern zu einer einzigen Anweisung zusammen.

- Durch einen Block werden **Sichtbarkeit** und **Gültigkeitsbereich** von Variablen begrenzt:

Lokale Variablen sind nur innerhalb des umfassenden Blocks gültig und sichtbar.

- In Java sind auch geschachtelte Mehrfachdeklarationen von Variablen gleichen Namens verboten: Lokale Variablen in inneren Blocks schränken die Sichtbarkeit von weiter außen definierten lokalen Variablen **nicht** ein; d.h. sie verursachen **keine „Verschattungen“** .

M. Wirsing: Einfache Rechenstrukturen und Kontrollfluß

Gültigkeitsbereich

- Der **Gültigkeitsbereich** einer lokalen Variablen oder Konstante ist **der die Deklaration umfassende Block**
- Außerhalb dieses Blocks existiert die Variable *nicht!*

Beispiel:

```
1. {
  int wert = 0;
  wert = wert + 17;
1.1 {
  int total = 100;
  wert = wert - total;
}
  wert = 2 * wert;
}
```

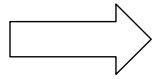
} Block 1.1
Gült.ber.
total

} Block 1.
Gült.ber.
wert

M. Wirsing: Einfache Rechenstrukturen und Kontrollfluß

Pulsierender Speicher

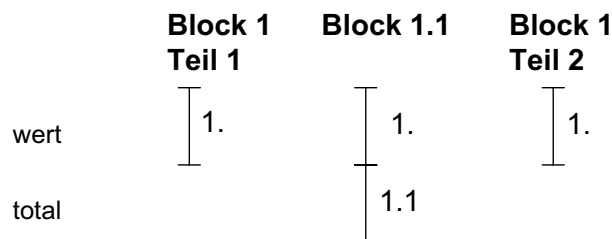
- Die Menge der gültigen lokalen Variablen verändert sich **kellerartig** mit jedem Eintritt und Austritt aus einem Block:
Bei Eintritt kommen neue Variablendeklarationen (als letzte) hinzu, die beim Austritt (als erste) wieder ungültig werden.



Pulsierender Speicher, implementiert durch Laufzeitkeller

Beispiel:

Abarbeitung von



Fallunterscheidung

Die Fallunterscheidung in Java hat die Form

```
if ( <Bool Expression> ) <Statement>
```

bzw.

```
if ( <Bool Expression> ) <Statement> else <Statement>
```

Beispiel: Kontofluß 1

```
if (kontoStand >= betrag)
    kontoStand = kontoStand - betrag;
```

Beispiel: Kontofluß 2

```
if (kontoStand >= betrag)
    kontoStand = kontoStand - betrag;
else
    kontoStand = kontoStand - betrag - UEBERZIEH_GEBUEHR;
```

Mehrfache Anweisungen im `else`-Zweig

- Blockklammern sind bei geschachtelten Fallunterscheidungen und im `else` Zweig sehr wichtig:
Vergessen führt zu falschen Ergebnissen

Beispiel:

```
if (kontoStand >= betrag)
{
    double neuerStand = kontoStand - betrag;
    kontoStand = neuerStand;
}
else
    kontoStand = kontoStand - betrag - UEBERZIEH_GEBUEHR;
    gebuehren = gebuehren + UEBERZIEH_GEBUEHR;
```

Was ist falsch?

M. Wirsing: Einfache Rechenstrukturen und Kontrollfluß

Korrektur des Beispiels

Beispiel:

```
if (kontoStand >= betrag)
{
    double neuerStand = kontoStand - betrag;
    kontoStand = neuerStand;
}
else
{
    kontoStand = kontoStand - betrag - UEBERZIEH_GEBUEHR;
    gebuehren = gebuehren + UEBERZIEH_GEBUEHR;
}
```

Überziehungsgebühr nur, wenn
Konto nicht gedeckt!

M. Wirsing: Einfache Rechenstrukturen und Kontrollfluß

Iteration

Drei Konstrukte zur Iteration:

- while
- for
- (do In der Vorlesung nicht betrachtet)

While-Schleifen

Die **while**-Schleife hat die Form

```
while (<Boolescher Ausdruck>) <Statement>
```

Beispiel 1

```
int n = 1, end = 10;  
while (n <= end)  
{  
    System.out.println(„tick“ + n);  
    n++;  
}
```

Beispiel 2

```
int qs = 0, x = 352;  
while (x > 0)  
{  
    qs = qs + x % 10;  
    x = x/10;  
}
```

for- Schleifen

- Die häufigste Form der while-Schleife ist

```

int i = start;    // Initialisierung
while (i <= end) // Bedingung
{
    ...
    i++;           // Zählerkorrektur durch
                  // konstante Änderung
}

```

- wird abgekürzt durch

```

for (int i = start; i <= end; i++)
{
    ...
}

```

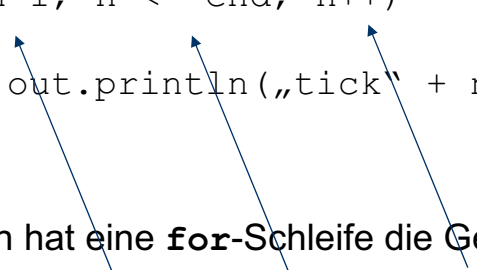
for-Schleifen

Beispiel:

```

int end = 10;
for (int n=1; n <= end; n++)
{
    System.out.println("tick" + n);
}

```



- Allgemein hat eine **for**-Schleife die Gestalt

for (Initialisierung; Bedingung; Zählerkorrektur) <Statement>

- Dabei wird zunächst die Initialisierung ausgeführt.

Dann wird, solange die Bedingung wahr ist, <Statement> ausgeführt und der Zähler geändert (gemäß der Zählerkorrektur-<expression>).

for-Schleifen

- Guter Stil ist es, **for**-Schleifen nur folgendermaßen zu schreiben:

```
for (setze counter auf start;  
      Test, ob counter bei end;  
      aendere counter)  
{  
    ...//counter, start, end und increment werden  
    //hier nicht geändert!  
}
```

- Außerdem sollte der Zähler **counter** in der Schleifen-Initialisierung deklariert werden.

Zusammenfassung 1

- Java besitzt
 - 4 Grunddatentypen für ganze Zahlen (**byte, short, int, long**) und
 - 2 Grunddatentypen für Gleitpunktzahlen (**float, double**).
- Dazu kommen noch **boolean** und **char**.
- String ist **kein** Grunddatentyp.
- Java hat eine **automatische Konversion in den allgemeineren** Grunddatentyp.
- Konversion in einen **spezielleren Datentyp** geschieht explizit durch **Typcasting**.

Zusammenfassung 2

- Grundlegende **imperative Konstrukte** von Java sind:
 - Deklaration lokaler Variablen, Zuweisung, Sequ. Komposition,
 - Fallunterscheidung, Iteration
- Lokale Variablen müssen **vor Benutzung initialisiert** werden und werden im **Keller** gespeichert.
- Eine Fallunterscheidung erlaubt es, abhängig von einer Bedingung, verschiedene Anweisungen auszuführen.
- while-Schleifen bilden die Grundform der Iteration;
for-Schleifen sollten verwendet werden, wenn die Schleifenvariable von einem Anfangswert bis zu einem Endwert mit einem **konstanten Inkrement** oder Dekrement läuft;