Einfache Rechenstrukturen und Kontrollfluss

Martin Wirsing

in Zusammenarbeit mit
Michael Barth, Fabian Birzele und Gefei Zhang

http://www.pst.informatik.uni-muenchen.de/lehre/WS0506/infoeinf/

WS 05/06

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

Grunddatentypen in Java

- Ganze Zahlen
- Gleitpunktzahlen
- Zeichen
- Boole'sche Werte
- und Felder (später)

Ganze Zahlen

- `byte` 1 Byte -128 bis 127
  -2^7 bis 2^7-1
- `short` 2 Byte -32768 bis 32767
- `int` 4 Byte -2,147,483,648 bis 2,147,483,647
  -2^{31} bis $2^{31}-1$
- `long` 8 Byte
  -9,223,372,036,854,775,808 bis 9,223,372,036,854,775,807

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:

`1 + 1.7`	ist vom Typ **`double`**
`1.0f + 1`	ist vom Typ **`float`**
`1.0f + 1.0`	ist vom Typ **`double`**

Typkonversion

Type Casting:

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

Beispiele:

(**`byte`**) `3`	ist vom Typ **`byte`**
(**`int`**)`(2.0 + 5.0)`	ist vom Typ **`int`**
(**`float`**)`1.3e-7`	ist vom Typ **`float`**

Bei der Typkonversion kann Information verloren gehen.

Beispiel:

```
(int) 5.2 ==  5
(int)-5.2 == -5
```

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“

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“)
&&	sequentielle Konjunktion
||	sequentielle Adjunktion

op strikt bedeutet, dass der Wert von `x op y` undefiniert ist, falls der Wert von x **oder** der Wert von y undefiniert ist, z.B. `(false & undef) == undef`.

op sequentiell bedeutet, dass `x op y` von links nach rechts ausgewertet wird und die Undefiniertheit von y keine Rolle spielt, wenn der Wert von `x op y` schon “klar“ ist. Bsp. `(false && undef) == false`.

Boole‘sche Werte

„entweder-oder“

^	true	false
true	false	true
false	true	false

true ^ false == true
true ^ true == false

und

„oder“

|,||	true	false
true	true	true
false	true	false

true | false == true
true | true == true

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.

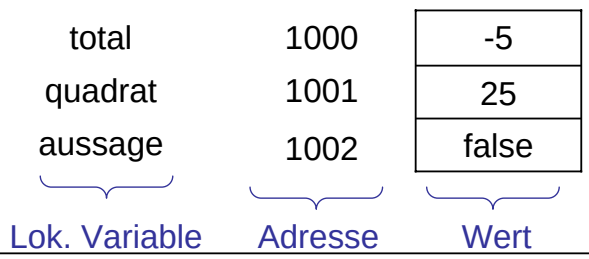
Zustand

- 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:



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;
```


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`

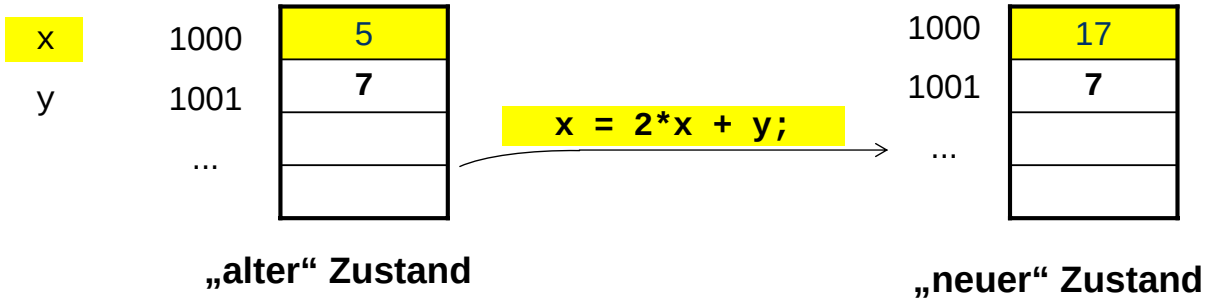
Zuweisung

Bei der **Zuweisung**

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

wird der Wert w der `<Expression>` im „alten“ Zustand berechnet und im Nachfolgezustand der Variablen `<VarName>` als neuer Wert zugewiesen.

Beispiel:



Zuweisung: Textuelle Darstellung

Beispiel textuell:

„alter“ Zustand `s1 = [(x, 5), (y,7), (b, true)]`

Zuweisung `x = 2*x+y;`

„neuer“ Zustand `s2 = [(x, 17), (y,7), (b, true)]`

Zuweisung: Abkürzende Schreibweisen

Abkürzungen

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

Beispiele

`x += y;`	steht für `x = x + y;`
`b &&= c;`	steht für `b = b && c;`
`x += 3*y;`	steht für `x = x + 3*y;`

Sequentielle Komposition

Sequentielle 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“** .

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`

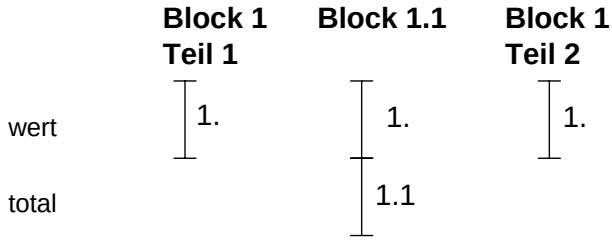
Block 1. Gült.ber. `wert`

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?

Korrektur des Beispiels

Beispiel:

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

Überziehgsgebühr nur, wenn Konto nicht gedeckt!

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
}
```

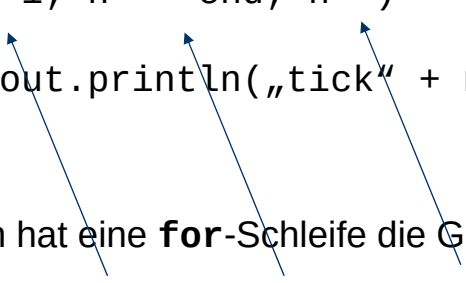
- wird abgekürzt durch

```
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 geaendert!
}
```

- 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;