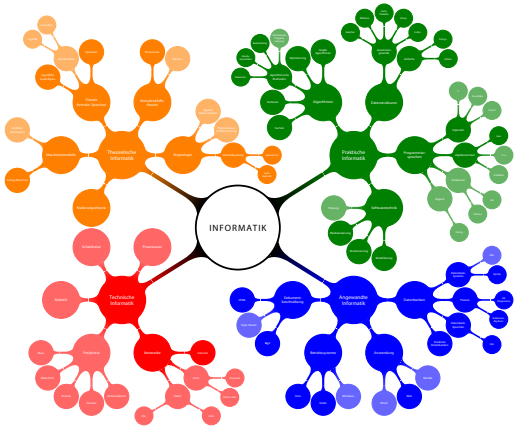
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 11

Steuerungsanweisungen und elementate Datentypen

Das Kleine-Ein-mal-Eins der Programmierung

Vorlesung Einführung in die Informatik 1 vom 28. November 2013 von Till Tantau



Lernziele von Kapitel 11

1. Die Steuerungsanweisungen If-Then-Else, While-Schleife und For-Schleife benutzen können
2. Variablendeklarationen verstehen und benutzen können
3. Elementare Datentypen von Java kennen
4. Typkorrektheit und Typfehler verstehen

Gliederung von Kapitel 11

Die Alternativen-Steuerungsanweisung.

- Die Alternative besteht aus zwei bis drei Teilen:
  1. Einer *Bedingung*.
  2. Einem *Then-Zweig*.
  3. Einem *Else-Zweig* (optional).
- Die Bedingung ist ein boolescher Ausdruck. Dieser wird als erstes ausgewertet:
  - Wenn er wahr ist, werden die Anweisungen im Then-Zweig befolgt.
  - Wenn er falsch ist, werden die Anweisungen im Else-Zweig befolgt. (Falls dieser fehlt, geht es einfach mit dem nächsten Befehl weiter.)

Schreibweise der Alternative in Java.


```
if (Bedingung) {
  Then-Zweig
}
else {
  Else-Zweig
}
```

- Sind Then-Zweig oder Else-Zweig *einzelne Anweisungen*, so kann man die geschweiften Klammern weglassen.
- Im Else-Zweig darf eine »einzelne Anweisung« auch wieder eine Steuerungsanweisung sein.
- Dann wird es aber unübersichtlich.

Der Algorithmus mit If-Then-Else in Java-Syntax.

Spezifikation

Bestimme Durchschnitt und Median der Zahlen *a*, *b* und *c*.

Algorithmus

```
average = (a + b + c) / 3;
if ((a <= b && a >= c) || (a >= b && a <= c)) {
  median = a;
}
else {
  if ((b <= a && b >= c) || (b >= a && b <= c)) {
    median = b;
  }
  else {
    median = c;
  }
}
```

Die While-Steuerungsanweisung erlaubt es, eine Befehlsfolge wiederholt auszuführen.

- Die While-Schleife besteht aus zwei Teilen:
  1. Einer *Bedingung*.
  2. Einem *Körper*.
- Die Bedingung ist ein boolescher Ausdruck. Dieser wird bei jedem Schleifendurchlauf am Anfang ausgewertet.
- Wenn die Bedingung wahr ist, wird der Körper ausgewertet, danach wieder die Bedingung, dann wieder der Körper und so fort.
- Wenn die Bedingung falsch ist, wird hinter dem Körper mit dem nächsten Befehl weitergemacht.

Die Syntax der While-Schleife ist einfach.


```
while (Bedingung) {
  Körper
}
```

Wieder kann man die geschweiften Klammern weglassen, wenn der Körper eine einzelne Anweisung ist.

5-Minuten-Aufgabe

Zur Übung

Welchen Wert haben `sum` und `n` am Ende von folgendem Programm?

```
int n = 0;
int sum = 0;
while (n <= 100) {
  sum = sum + n;
  n = n + 1;
}
```

Wem das zu leicht ist: Welchen Wert hat `sum` am Ende, wenn die Bedingung `sum <= 100` lautet?

Die For-Steuerungsanweisung ist eine bequeme While-Schleife.

- Die For-Schleife besteht aus vier Teilen:
  1. Einer *Initialisierung*,
  2. einer *Bedingung*,
  3. einem *Inkrement* und
  4. einem *Körper*.
- Am Anfang wird die Initialisierung ausgeführt.
- Dann wird die Bedingung getestet. Solange sie wahr ist, wird der Körper ausgeführt.
- Jedesmal, nachdem der Körper ausgeführt wurde, wird das Inkrement ausgeführt.

Die Syntax der For-Schleife.


```
for (Initialisierung; Bedingung; Inkrement) {
  Körper
}
```

Wieder kann man die geschweiften Klammern weglassen, wenn der Körper eine einzelne Anweisung ist.

Die folgenden Programme haben den gleichen Effekt.


```
int n = 0;
int sum = 0;
while (n <= 1000) {
  sum = sum + n;
  n = n + 1;
}
...
```

```
int n;
int sum = 0;
for (n = 0; n <= 1000; n = n + 1) {
  sum = sum + n;
}

...
```

5-Minuten-Aufgabe

Zur Übung

Folgender Algorithmus druckt alle Teiler einer Zahl *n* aus:

Algorithmus

Für alle Zahlen *i* zwischen 1 und *n* tue
    Falls *n* modulo *i* gleich 0 ist
        Gib *i* aus

Formulieren Sie den Algorithmus als Java-Programm. Zur Ausgabe eine Zahl können Sie `System.out.println(i)` verwenden.

Daten haben natürlicherweise einen Typ.

Daten lassen sich leicht in Klassen von *ähnlichen Daten* einteilen.
Beispiele sind:

- Zahlen (in verschiedenen Varianten),
- Texte,
- Bilder,
- Dateien,
- Daten,
- Koordinatenpunkte.

Solche Klassen nennen wir einen *Typ*:

- Ein *Datentyp* ist eine Menge von möglichen Werten.
- Hat beispielsweise ein Wert den Datentyp `int`, so weiß man, dass dieser Wert eine ganze Zahl ist.

Wozu dienen Datentypen?

- Datentypen helfen *Menschen*,
  Ordnung in ein Programm zu bringen.
- Datentypen helfen *Übersetzern*,
  die richtige Menge Speicher für einen Wert zu reservieren.
- Datentypen helfen *Menschen und Übersetzern*,
  Fehler zu finden: Man kann ein Bild nicht quadrieren.

Die Typen von Typen.

Es gibt viele unterschiedliche Typen.
Man kann sie grob in zwei Klassen einteilen:

- *Elementare Typen*
  Diese einfachen Datentypen sind fest eingebaut in den Übersetzer. Sie haben keine innere Struktur.
  Beispiel: `int`
- *Zusammengesetzte Typen*
  Solche Datentypen haben eine innere Struktur. Diese Datentypen können (und müssen) im Programm selbst erzeugt werden.
  Beispiel: Ein Datumstyp. Er besteht aus drei Zahlen.

Liste der elementaren Datentypen von Java.

Datentyp	Bits	Wertebereich
`boolean`	1	`true` und `false`
`byte`	8	−128 bis 127
`short`	16	−32768 bis 32767
`int`	32	ca −2.000.000.000 bis 2.000.000.000
`long`	64	−ganz viel bis ganz viel
`float`	32	ca. 8 Dezimalstellen Genauigkeit
`double`	64	ca. 16 Dezimalstellen Genauigkeit
`char`	16	ein Unicode Zeichen wie `'x'`

Das Konzept der Typisierung.

- In *stark typisierten Programmiersprachen* hat jeder Wert, jeder Ausdruck und jede Variable einen bestimmten Typ.
- Durch die Typisierung weiß der Übersetzer immer,
  - wie viel Speicher für die Variablen und Ausdrücke benötigt wird und
  - ob die Ausdrücke überhaupt »typkorrekt« sind.
- Wir wissen schon: In Java muss für jede Variable *vor ihrer ersten Benutzung* der Typ dem Übersetzer mitgeteilt werden.
  Dazu schreibt man `type var;` um anzudeuten, dass eine Variable `var` den Typ `type` haben soll.

Eine Bequemlichkeit in der Syntax.

Java erlaubt es,

- mehrere Variablen des gleichen Typs in einer Deklaration zu deklarieren und
- ihnen auch gleich einen Wert geben.

Beispiel

Die folgenden Programmtexte haben den gleichen Effekt:

1. ```
   int a = 5, b = 6, c;
   ```
2. ```
   int a;
   a = 5;
   int b;
   b = 6;
   int c;
   ```

5-Minuten-Aufgabe

Zur Übung

Welche der folgenden Deklarationen sind korrekt?

1. ```
   char
   c = 'a';
   ```
2. ```
   char a,b;
   ```
3. ```
   int int;
   ```
4. ```
   int i, char j;
   ```
5. ```
   double x = 5;
   ```
6. ```
   boolean flag = 'a';
   ```
7. ```
   boolean flag = true;
   ```
8. ```
   boolean flag == y;
   ```
9. ```
   boolean flag = y == 5;
   ```
   ```

Jeder Ausdruck hat einen Typ.

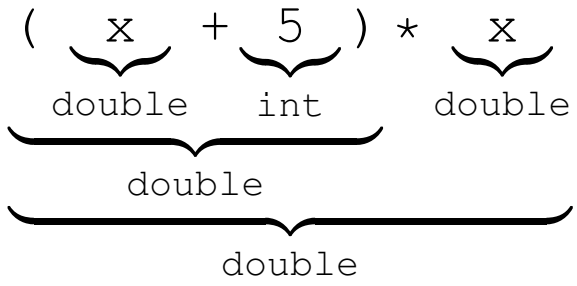
Zur Erinnerung: *Ausdruck* ist

- eine Variable oder
- ein Wert (wie eine Zahl) oder
- die Verknüpfung von Ausdrücken mittels eines Operators (wie `x + 5`).

In Java haben alle Ausdrücke und auch alle Teilausdrücke einen Typ.

Beispiel

Sei `x` als `double` deklariert (also `double x;`).

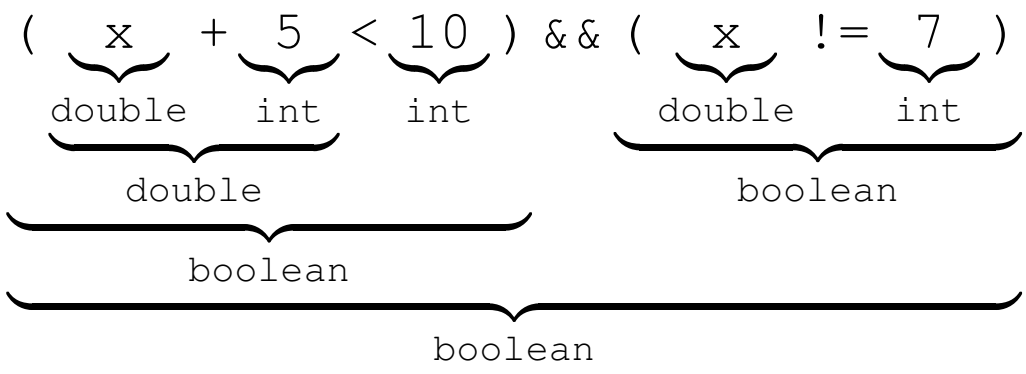


Wie bekommt der Ausdruck seinen Typ?

Die Typen der Teilausdrücke werden automatisch bestimmt.

Beispiel

Sei `x` als **`double`** deklariert.



Die zwei Arten von Fehlern in Programmen.

Es gibt zwei Hauptarten von Programmfehlern:

- Fehler zur Laufzeit (runtime errors) und
- Fehler zur Übersetzungszeit (compile time errors).

Beispiel (Runtime error)

- Ein Programm liest einen Wert in die Variable `x` von der Tastatur.
- Dann gibt es die Zuweisung `y = 5 / x;`.
- Falls nun eine Null gelesen wurde, so stürzt das Programm *jetzt* ab.

Beispiel (Compile time error)

```
x = 5 +;
```

Typfehler sind Compile-time-errors.

- Ein *Typfehler* liegt vor, wenn man versucht, einer Variable einen Wert zuzuweisen, der den falschen Typ hat.
  Beispiel: `double x = true;`
- Ein *Typfehler* liegt auch vor, wenn man einen Operator falsch anwendet.
  Beispiel: `5 && x < 0`
- Ein *Typfehler* liegt auch vor, wenn ein anderer Typ als **`boolean`** in einer Bedingung benutzt wird.
  Beispiel: `if (42.4)`
- Die Erkennung von Typfehlern durch den Übersetzer ist eine große Hilfe bei der Erstellung großer Programme. Deshalb werden in großen Projekten in der Regel stark typisierte Sprachen verwendet.

Zusammenfassung

Java-Syntax: Die elementaren Datentypen

- Die elementaren Typen in Java sind `boolean`, `byte`, `short`, `int`, `long`, `char`, `float`, `double`.
- Jeder Ausdruck, jeder Wert, jede Variable hat in Java einen bestimmten Typ.
- Werden Typen falsch verwendet, so liegt ein *Typfehler* vor, was *während der Übersetzung* festgestellt wird.

Zusammenfassung

Java-Syntax: Die Alternative

```
if (Bedingung) {
  Then-Zweig
}
else {
  Else-Zweig
}
```

Java-Syntax: Die While-Schleife

```
while (Bedingung) {
  Körper
}
```

Java-Syntax: Die For-Schleife

```
for (Initialisierung; Bedingung; Inkrement) {
  Körper
}
```