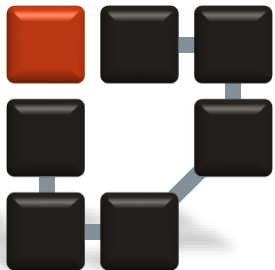


Informatik 1 für Nebenfachstudierende Grundmodul

Java – Lexikalische Struktur

Kai-Steffen Hielscher
Folienversion: 10. Dezember 2018



Informatik 7
Rechnernetze und
Kommunikationssysteme



**FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG**
TECHNISCHE FAKULTÄT

Inhaltsübersicht

- Kapitel 3 - Java
 - Einführung
 - **Lexikalische Struktur**
 - Datentypen und Variablen

Literaturhinweise

- RATZ, SCHEFFLER, SEESE, WIESENBERGER:
Grundkurs Programmieren in Java,
7. Auflage, Hanser, München 2014.
- LOIS, MÜLLER:
Java – Eine Einführung in die Programmierung,
Hanser, München 2014.
- Oracle Java Tutorials Learning Paths
<https://docs.oracle.com/javase/tutorial/tutorialLearningPaths.html>
- viele weitere gute Quellen und Tutorials online

Lexikalische Struktur

- Groß-/Kleinschreibung
- Anweisungen und Semikola
- Whitespace und Zeilenumbrüche
- Kommentare
- Literale
- Bezeichner (Identifizier)
- Schlüsselwörter

Lexikalische Struktur

- Java stark beeinflusst von C und C++ (objektorientierte Erweiterung der Sprache C)
- Lexikalische Struktur
 - Menge von Regeln
 - bestimmt, wie Programmtexte aufgebaut sein müssen
- Niedrigste Ebene der Sprachsyntax beschreibt u.a.
 - Aussehen von Variablen- und Funktionsnamen
 - Aufbau von Kommentaren
 - Trennung von Programmanweisungen

} Syntax der
Sprache

Groß-/Kleinschreibung

- Java unterscheidet stets zwischen Groß- und Kleinschreibung
 - die Bezeichner `a` und `A` kennzeichnen unterschiedliche Dinge
 - ebenso sind `Hallo`, `hallo`, `hALLO`, `hAllo` und `HALLLO` verschieden
 - **Vorsicht beim Programmieren, kann zu Fehlern führen.**

Anweisungen und Semikola

- Anweisungen werden in Java immer mit einem Semikolon ; abgeschlossen, das Semikolon trennt also die einzelnen Anweisungen voneinander
 - entspricht in etwa dem Punkt in der deutschen Sprache
 - Anweisung: einzelne Wertzuweisung, Methodenaufruf, Schleife, ...
 - Beispiel:

```
a = 5;  
System.out.println("Das ist meine Ausgabe.");  
g = ggT(21, 35);  
kgV = (m * n) / ggT(m, n);
```

Anweisungen und Semikola

- Verbundanweisungen (compound statement)
 - Folge von Anweisungen bildet einen Block, der von geschweiften Klammern { } eingeschlossen wird
 - kein Semikolon nach der schließenden geschweiften Klammer

- Beispiel:

```
if(z < 3) {  
    kleiner = true;  
    System.out.println("Die Zahl z ist wirklich");  
    System.out.println("kleiner als 3.");  
}
```


Whitespace und Zeilenumbrüche

- Whitespace (Zeilenumbrüche, Leerzeichen, Wagenrücklauf, Zeilenschaltung, Tabulatoren) werden von Java ignoriert.
- Nutzung zur optischen Gliederung des Quelltextes
- Beispiel

```
preisstaeigerung ( inventar, inflation, lebensunterhalt, gier );
```

ist äquivalent zu

```
preisstaeigerung (  
    inventar,  
    inflation,  
    lebensunterhalt,  
    gier  
);
```

oder

```
preisstaeigerung(inventar,inflation,lebensunterhalt,gier);
```

Whitespace und Zeilenumbrüche

- Code durch Einrückung leserlicher machen
 - Ausrichten gleichartiger Dinge
 - verschachtelte Anweisungen
 - lange Ausdrücke
 - verkettete Zeichenketten
 - Blockstruktur zusätzlich zu geschweiften Klammern durch Einrückung kennzeichnen
- **Verzicht auf optische Gliederung macht Programme unlesbar.**

Kommentare

- Kommentare informieren Menschen, werden von Java ignoriert
- immer kommentieren:
 - sehr Abstraktes (z.B. reguläre Ausdrücke)
 - geniales (kurzer Code mit weitreichender Funktionalität)
 - Code mit unerwarteten Nebenwirkungen
- Offensichtliches nicht kommentieren
 - Wirkung arithmetischer Ausdrücke $a = b + c$; (trivial)
 - einfache, lokal begrenzte Aktionen

Kommentare

- Bezeichner von Variablen, Methoden und Klassen dienen auch der Kommentierung

- lange, aussagekräftige Namen für Dinge mit weitreichenden Konsequenzen, z.B.

```
anzahlBildElemente = 5;
```

- kurze Namen erhöhen Übersicht bei lokal begrenzten Aufgaben, z.B. Schleifen:

```
for(i = 1; i < 7; i++) {  
    System.out.println("counter: " + i);  
}
```

Kommentare

■ Zeilenkommentare

- beginnen mit den Zeichen `//` und enden mit dem Zeilenende
- alle Zeichen nach den Kommentarzeichen werden von Java ignoriert
- Beispiel:

```
kgV = (m * n) / ggT(m, n); // kleinstes gemeinsames Vielfaches
System.out.println("kgV(" + m + ", " + n + ") = " + kgV);
```

■ Blockkommentare

- soll ein Kommentar mehrere Zeilen umfassen, muss mit Zeilenkommentaren jede Zeile mit `//` beginnen
- Blockkommentare beginnen mit `/*` und enden mit `*/`
Blockkommentare können mehrere Zeilen umfassen
- Beispiel:

```
/* Diese Funktion berechnet das kleinste gemeinsame Vielfache
   zweier Zahlen aus deren größtem gemeinsamen Teiler, siehe:
   https://de.wikipedia.org/wiki/Kleinstes_gemeinsames_Vielfaches
*/
```

Kommentare

■ JavaDoc-Format

- erlauben automatische Generierung einer Dokumentation
- beginnen mit `/**` und enden mit `*/`
- üblicherweise beginnt jede Zeile dazwischen mit `*`, ist aber nicht zwingend erforderlich
- zunächst allgemeine Beschreibung, die auch ohne den Quellcode verstanden werden kann
- dann unterschiedliche Kommentarbefehle, die jeweils mit `@` beginnen, z.B.
 - `@author`
 - `@version`

Kommentare

■ JavaDoc-Format

■ Beispiel

```
/**  
 * Dieses Programm berechnet das kleinste gemeinsame  
 * Vielfache zweier Zahlen aus deren größtem  
 * gemeinsamen Teiler  
 *  
 * @author Euklid von Alexandria  
 * @version 1.2  
 */
```

Literale

- Literale legen einen konstanten Wert fest
 - werden vom Compiler in interne Repräsentation überführt und in Java Bytecode übernommen
 - verschiedene Typen
 - ganze Zahlen, z.B. `4711` oder `-28`
 - Oktalzahlen mit führender `0`, z.B. `077`
 - Hexadezimalzahlen beginnen mit `0x`, z.B. `0xBE` oder `0xefef`
 - Binärzahlen mit `0b`, z.B. `0b1001`
 - Gleitkommazahlen mit `.`, z.B. `3.14159265`
 - Exponent mit `e`, z.B. `6.022e23` entspricht $6,022 \cdot 10^{23}$
 - Wahrheitswerte, `true` und `false`
 - einzelne Zeichen in einfachen Anführungszeichen `'`, z.B. `'a'`
 - Sonderzeichen, z.B. `'\n'`, `'\t'`, ...
 - Zeichenketten in Anführungszeichen, z.B. `"Hallo"`
 - Null-Literal für Referenzen, `null`

Bezeichner

- verschiedene Dinge in Programmen brauchen einen Namen, beispielsweise Variablen, Klassen und Methoden, Programmierer gibt einen Bezeichner (Namen) vor
- auch hierbei ist die Groß-/Kleinschreibung relevant, `a` ist eine andere Variable als `A`
- erlaubte Zeichen für Bezeichner
 - **Kleinbuchstaben** `a ... z`
 - **Großbuchstaben** `A ... Z`
 - auch **internationale Buchstabenzeichen** (`ä, ß, μ, 力, Ж, ...`) sind möglich, da Unicode unterstützt wird, sollten aber vermieden werden
 - **Ziffern** `0 ... 9`
 - **Unterstrich** `_`
 - **Dollarzeichen** `$`
 - Bezeichner dürfen **nicht mit einer Ziffer beginnen**
 - **reservierte Schlüsselwörter** dürfen nicht als Bezeichner verwendet werden (`if, else, for, ...`), siehe spätere Folien

Schlüsselwörter

- bestimmte Wörter dürfen nicht als Bezeichner gewählt werden
 - Literale `true`, `false` und `null`
 - reservierte Wörter, **Schlüsselwörter** mit besonderer Bedeutung in Java

Schlüsselwörter

<code>abstract</code>	<code>assert</code>	<code>boolean</code>	<code>break</code>	<code>byte</code>
<code>case</code>	<code>catch</code>	<code>char</code>	<code>class</code>	<code>const</code>
<code>continue</code>	<code>default</code>	<code>do</code>	<code>double</code>	<code>else</code>
<code>enum</code>	<code>extends</code>	<code>final</code>	<code>finally</code>	<code>float</code>
<code>for</code>	<code>goto</code>	<code>if</code>	<code>implements</code>	<code>import</code>
<code>instanceof</code>	<code>int</code>	<code>interface</code>	<code>long</code>	<code>native</code>
<code>new</code>	<code>package</code>	<code>private</code>	<code>protected</code>	<code>public</code>
<code>return</code>	<code>short</code>	<code>static</code>	<code>strictfp</code>	<code>super</code>
<code>switch</code>	<code>synchronized</code>	<code>this</code>	<code>throw</code>	<code>throws</code>
<code>transient</code>	<code>try</code>	<code>void</code>	<code>volatile</code>	<code>while</code>