Java-Referenz

Diese Referenz stellt nur einen kleinen Auszug der gesamten Java-Syntax dar.

Java-Grundgerüst

Das Java-Grundgerüst wird von jedem Java-Programm benötigt. Die Java-Befehle müssen innerhalb der geschweiften Klammern eingefügt werden.

Beispiel:

```
public class Klassenname {
    public static void main (String args[]) {
    ...
    }
}
```

Importieren von Bibliotheken

Mit der import-Anweisung können Programme aus anderen Bibliotheken eingebunden werden. Die `import`-Anweisung muss vor dem Grundgerüst stehen.

Beispiel:

```
// Import der Bibliotheken java.util und java.io
    import java.util.*;
    import java.io.*;
```

Datentypen von Variablen

Damit der Computer weiß, welche Bedeutung ein Speicherinhalt hat, muss einer Variablen ein Datentyp zugewiesen werden. Die Tabelle zeigt nur einige wichtige Datentypen.

Bei dem Datentyp String handelt es sich nicht um einen elementaren Datentyp, sondern um eine Klasse. `String`-Objekte brauchen allerdings nicht mit `new` erzeugt werden.

Datentyp	Bedeutung	Größe in Bytes
`int`	Ganze Zahl (Integer)	4
`double`	Dezimalzahl	8
`String`	Text	beliebig
`boolean`	Wahrheitswert	1
`char`	Einzelnes Tastaturzeichen	2

Deklaration von Variablen

Durch die Deklaration werden einer Variablen ein Name und ein Speicherplatz zugewiesen. In Java müssen alle verwendeten Variablen deklariert werden.

Beispiele:

```
int ganzeZahl;
double dezimalZahl;
```

	`boolean wahrheitsWert;`
	`String text;`
	`char einzelnesZeichen;`

Zuweisungen	Mit Zuweisungen werden Variablen Werte zugewiesen. Der Wert muss mit dem Datentyp der Variablen übereinstimmen.
	Allen Variablen, die auf der rechten Seite einer Zuweisung stehen, muss ein Wert zugewiesen worden sein (Initialisierung). Die Initialisierung kann bereits bei der Deklaration erfolgen.
	Beispiele:
	`// Initialisierung`
	`int ganzeZahl = 0;`
	`double dezimalZahl = 0.0;`
	`String text = "Hallo";`
	`// Spätere Zuweisung im Code`
	`ganzeZahl = ganzeZahl + 1;`
	`dezimalZahl = 3.5;`
	`text = text + " Welt";`
	`zeichen = 'A';`
	`wahrheitsWert = true;`

Eingabebefehle	Bei Eingabe über die Konsole mit der Klasse `Scanner` muss zunächst ein Scanner-Objekt mit `new` erzeugt werden. Die Werte werden anschließend mit einer Methode wie z.B. `next()` eingelesen. Die Methode muss dem Datentyp der Variablen entsprechen, in der der Wert gespeichert wird.
	Beispiel:
	`/* Erzeugen eines Scanner-Objekts`
	`mit dem Varialennamen konsole */`
	`Scanner konsole = new Scanner(System.in);`
	
	`// Einlesen von Werten verschiedener Datentypen`
	`wort = konsole.next();`
	`ganzeZahl = konsole.nextInt();`
	`dezimalZahl = konsole.nextDouble();`

Ausgabebefehle	Die Ausgabe auf der Konsole erfolgt mit `System.out.println()`. Text und Werte von Variablen können zusammen ausgegeben werden. Beispiel: `System.out.println("Der Wert von x ist " + x);`
Verzweigungen	Mit der `if`-Anweisung können Befehle in Abhängigkeit von einer Bedingung ausgeführt werden. Die Befehle innerhalb der optionalen `else`-Klausel werden ausgeführt, wenn die Bedingung nicht erfüllt ist.

<table>
<tr><td></td><td>Mithilfe der <code>else if</code>-Klausel können Verzweigungen verschachtelt werden. In solchen Fällen wird nur die erste Anweisung ausgeführt, deren Bedingung erfüllt ist.
Beispiel:
<pre>if (a == 0) x = 0;
else if (a >= 10) x = 1;
else if (a >= 100) x = 2;
else if (a >= 1000) x = 3;
else x = -1;</pre></td></tr>
<tr><td>Schleifen</td><td>Schleifen dienen zur Wiederholung von Programmbefehlen. Man unterscheidet drei Arten von Schleifen:
• Die Zählschleife (<code>for</code>-Schleife)
• Die kopfgesteuerte Schleife (<code>while</code>-Schleife)
• Die fußgesteuerte Schleife (<code>do-while</code>-Schleife)
Beispiele:
<pre>// Zählschleifen (for-Schleife)
for (int i=0; i<=5; i++) x = x + 10;
// Kopfgesteuerte Schleifen (while-Schleife)
while (x < 0) x = x + 10;
// Fußgesteuerte Schleifen (do-while-Schleife)
do x = x + 10; while (x < 0);</pre></td></tr>
<tr><td>Bedingungen</td><td>Bedingungen treten in Kontrollstrukturen wie Schleifen oder Verzweigungen auf. Sie haben stets den Wahrheitswert <code>true</code> oder <code>false</code>. In Bedingungen kommen folgende Vergleichsoperatoren vor:
<table><tr><th>Operator</th><th>Bedeutung</th><th>Operator</th><th>Bedeutung</th></tr><tr><td>></td><td>größer</td><td>>=</td><td>größer gleich</td></tr><tr><td><</td><td>kleiner</td><td><=</td><td>kleiner gleich</td></tr><tr><td>==</td><td>gleich</td><td>!=</td><td>ungleich</td></tr></table>
Beim Vergleich von Objekten (d. h. auch bei Variablen vom Datentyp <code>String</code>) muss die Methode <code>equals</code> angewandt werden.
Weiterhin kommen die logischen Verknüpfungsoperatoren && (UND), || (ODER) und ! (NICHT) zur Anwendung.
Beispiele:
<pre>if (x != 0 || y == 0) ...;
while (x >= 100) ...;
do {...;} while (!eingabe.equals("STOP"));</pre></td></tr>
</table>

Mehrfach-verzweigungen	Mit Hilfe der *switch*-Anweisung können, ähnlich wie in einer *if-else*-Verzweigung, ein *int*-Wert überprüft und entsprechende Anweisungen ausgeführt werden.
	Beispiel:
	`switch (note)`
	`{`
	`case 1 : System.out.println("sehr gut"); break;`
	`case 2 : System.out.println("gut"); break;`
	`case 3 : System.out.println("befriedigend"); break;`
	`case 4 : System.out.println("ausreichend"); break;`
	`case 5 : System.out.println("mangelhaft"); break;`
	`case 6 : System.out.println("ungenügend"); break;`
	`default :`
	`System.out.println("Die Note muss zwischen 1 und 6 liegen!");`
	`}`

Methoden - Funktionen und Prozeduren	Programme können sinnvoll strukturiert und in einzelne Methoden (Funktionen und Prozeduren) aufgeteilt werden, sodass einer Klasse neben der *main*-Methode noch weitere Methoden zugeordnet sein können. Funktionen liefern im Gegensatz zu Prozeduren einen Rückgabewert. Beiden Arten von Methoden können mit Hilfe von Parametern Werte mitgegeben werden.
	Beispiel:
	`public static void main (String args[])`
	`{`
	`….`
	`double ergebnis, a, b;`
	`a=10;`
	`b=5;`
	`//Aufruf der Funktion summiere mit zwei double-Werten`
	`ergebnis = summiere (a, b);`
	`….`
	`}`
	`//Funktion summiere, die zwei Parameter vom double erwartet und`
	`//einen Rückgabewert vom Typ double liefert`
	`public static double summiere (double wert1, double wert2)`
	`{`
	`double summe;`
	`summe = wert1+wert2;`
	`return summe;`
	`}`

Besondere Techniken aus dem Unterricht I	*Lesen aus Dateien*
	Aus einer vorhandenen Datei `messdaten.txt` sollen Werte gelesen und weiterverarbeitet werden.
	`/* Das Lesen aus Dateien erfordert eine Fehlerbehandlung, die in der main-Methode zu definieren ist. */`
	`public static void main (String args[]) throws IOException`
	`// Erzeugen eines Datei-Objekts datei vom Typ Scanner`
	`Scanner datei = new Scanner(new File("messdaten.txt"));`
	`// Schleife zum Lesen des Datei-Inputs`
	`while (datei.hasNextLine()) {...;}`
	`/* Auslesen der nächsten Zeile der Datei in die Variable zeileninhalt vom Typ String und Kopieren in die Variable zeile vom Typ Scanner (zum weiteren Zerlegen). */`
	`zeileninhalt = datei.nextLine();`
	`Scanner zeile = new Scanner(zeileninhalt);`
	`/* Zugriff auf Einzelteile (Wort, Double-Wert, Int-Wert) der ausgelesenen Zeile mit Hilfe der Scanner-Methoden (next(), nextDouble(), nextInt()). */`
	`wort = zeile.next();`
	`zahlD= zeile.nextDouble();`
	`ZahlI= zeile.nextInt();`
	Erzeugen von Zufallszahlen
	JAVA stellt eine Klasse Random bereit. Mithilfe dieser Klasse ist es möglich, Zufallszahlen zu erzeugen. Die Klasse Random ist Bestandteil der Bibliothek java.util
	Mit der Klasse Random wird zunächst ein Zufallszahlenobjekt r erzeugt
	`Random r = new Random();`
	Die Variable r verfügt über eine Methode nextInt(), die eine ganze Zahl zurückgibt.
	`z = r.nextInt();`
	Das Ergebnis liegt aber im Bereich aller ganzen Zahlen, positive wie negative. Um eine Zufallszahl in einem bestimmten Bereich wie zwischen 1 und 6 zu würfeln, ist etwas Mathematik notwendig.
	`z = 1 + Math.abs(r.nextInt()) % 6;`
	Die Klasse Math wird ebenfalls von Java bereitgestellt. Deren Methode Math.abs() ermittelt den Absolutbetrag einer Zahl. Der Modulo-Operator % ermittelt den Rest einer Division. Bei einer Division durch 6 kann der Rest die Werte 0 bis 5 betragen. Daher wird noch eine 1 hinzuaddiert. Wir erhalten auf diese Weise eine Zufallszahl zwischen 1 und 6.

Besondere Techniken aus dem Unterricht II	*Zugriff auf eine MySQL-Datenbank*
	
	Der Zugriff auf eine MySQL-Datenbank wird mit Hilfe der Klasse *MySQLConnection* realisiert. Die Datei *MySQLConnection.java* ist deshalb in demselben Ordner zu speichern, in dem auch die auszuführende JAVA-Datei gespeichert ist.
	
	In der Klasse *MySQLConnection* werden fünf Datenbankangaben gespeichert, die ggf. anzupassen sind:
	
	1. Der Hostname lautet *localhost*, wenn die MySQL-Datenbank auf demselben Rechner liegt, wie das auszuführende JAVA-Programm. Anderenfalls wird hier die *IP-Adresse* des Rechners eingetragen, auf dem die Datenbank liegt.
	2. Der Standard-Port, über die die Kommunikation zur MySQL-Datenbank abgewickelt wird lautet *3306*. Hier sind in der Regel keine Anpassungen notwendig.
	3. Der Datenbankname muss der Datenbank entsprechen, auf die zugegriffen werden soll.
	4. Der Datenbankuser muss einem zulässigen Benutzer der MySQL-Datenbank entsprechen (z.B. *root*).
	5. Das Passwort des unter 4) angegebenen Benutzers muss hier angegeben werden (auch kein Passwort, d.h. “ “, ist hier möglich).
	
	In der auszuführenden JAVA-Datei, d.h. in ihrem Programm, müssen zwei Klassen importiert werden, damit der Zugriff mit Hilfe der Klasse *MySQLConnection* möglich ist.
	
	▪ import java.sql.ResultSet;
	▪ import java.sql.SQLException;
	
	Um in ihrem Programm eine SQL-Anweisung auf der MySQL-Datenbank auszuführen, wird diese zuerst in einer Variablen vom Typ *String*, hier *sqlString*, gespeichert.
	
	`String sqlString;`
	
	`sqlString = "SELECT vorname, nachname, alter FROM tbllehrer WHERE vorname = 'Max'"`
	
	Achtung: Innerhalb der SQL-Anweisungen müssen doppelte Hochkammata durch einfache Hochkommata ersetzt werden, damit im JAVA-Programm zwischen JAVA-String-Werten und SQL-Fixtext unterschieden werden kann.

SELECT-Anweisungen in JAVA-Programmen

Eine SELECT-Anweisung, hier in der Variablen *sqlString* enthalten, wird vom JAVA-Programm mit Hilfe der Anweisung M*ySQLConnection.getInstance().executeSQL(sqlString)* an die MySQL-Datenbank geschickt und dort ausgeführt. Das Ergebnis der SELECT-Anweisung wird zurückgeliefert und in einer Variablen vom Typ ResultSet, hier *result*, gespeichert.

```
ResultSet result =
MySQLConnection.getInstance().executeSQL(sqlString);
```

Die zurückgelieferten Datensätze können mit Hilfe der Methode next() zeilenweise, hier durch Verwendung einer while-Schleife, abgearbeitet werden. Auf die Felder eines Datensatzes kann dabei mit Hilfe der Methoden *getString()*, *getInt()*, *getDouble()* zugegriffen werden, um sie anschließend in Variablen, hier z.B. *var_vorname*, zu speichern.

```
while (result.next()) {
String var_vorname = result.getString("vorname");
String var_nachname = result.getString("nachname");
int var_alter = result.getInt("alter");
String ausgabe = var_vorname + ", " + var_nachname + ","
+ var_alter;
System.out.println(ausgabe);
}
```

INSERT-, UPDATE- und DELETE-Anweisungen in JAVA-Programmen

Eine INSERT-, UPDATE- oder DELETE-Anweisung, hier in der Variablen *sqlString* enthalten, wird vom JAVA-Programm mit Hilfe der Anweisung M*ySQLConnection.getInstance().executeIUD(sqlString)* an die MySQL-Datenbank geschickt und dort ausgeführt. Diese Anweisung liefert keine Ergebnisse zurück. Die Daten in der Datenbank werden entsprechend der Anweisung eingefügt, geändert oder gelöscht.

```
sqlString = "DELETE FROM tbllehrer WHERE nachname =
'Bölte'";
MySQLConnection.getInstance().executeIUD(sqlString);
```