Programmierung 2

Kapitel 2 – Datentypen und Operatoren

Vielen Dank an Prof. Erik Buchmann
für die beigesteuerten Folien

Unterschiede zwischen Visual Basic und Java

Visual Basic

- Variablen

```
Dim i, j, k As Integer
' Integer ... elem. Datentyp
```

- Felder

```
Dim zahlen(6) As Integer
' 7 Elem. (Index: 0..6)

i = zahlen(2)
' Zugriff auf 3.El. (Index 2)

Dim zahlen(2 To 6) As Integer
' 5 Elem. (Index: 2..6)
```

Java

```
int i, j, k;
// int ... elem. Datentyp
// Integer ... abstr. Datentyp (Klasse)
```

```
int zahlen[] = new int[7];
// 7 Elem. (Index: 0..6)

i = zahlen[2];
// Zugriff auf 3.El. (Index 2)

// geht in Java nicht!
```

Visual Basic

- Prozedur (kein Rückgabewert!)

```
Sub meldeMsg(ByVal msg As String)
    MsgBox(msg)
End Sub
```

- Funktion (mit Rückgabewert)

```
Function sqr(ByVal z As Double) As Double
    Return z*z
End Function

Function sqr(ByVal z As Double) As Double
    sqr = z*z
    ' damit wird der Rueckgabewert gesetzt,
    ' vor dem Verlassen kann weiterer Code folgen
End Function
```

Java

```
void meldeMsg(String msg) {
  System.out.println(msg);
}
```

```
double sqr(double sqr){
  return z*z;
}
```

Visual Basic

- Einfaches Programm

```
Module Program
  Sub Main(args As String())
    Console.WriteLine("Hello World!")
  End Sub
End Module
```

Java

```
public class Program {
  public static void main( String[] args) {
    System.out.println("Hello World!");
  }
}
```

Grundgerüst

```
package hftl;

/**
 * Diese Klasse ist ein Hallo-Welt
 * @author buchmann
 */
public class HelloWorld {
    /**
     * Dies ist die Startmethode eines Java-Programms
     * @param args Array mit Argumenten von der Kommandozeile
     */
    public static void main(String[] args) {
        // Gebe "Hallo Welt!" auf der Konsole aus
        System.out.println("Hallo Welt!");

        // Programmende
    }
}
```

Package
(für heute: Unterverzeichnis)

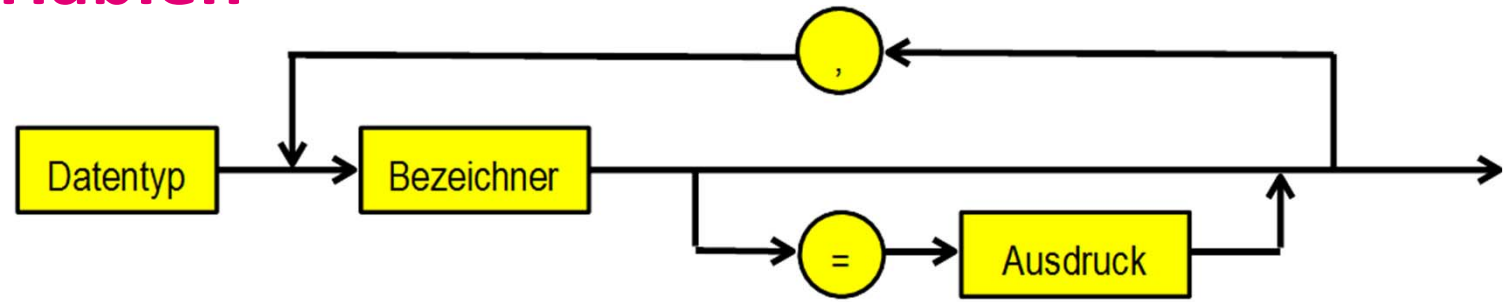
Klassenname
(für heute: Dateiname ohne Endung)

Übergabeparameter

Imperatives Programm

Einstiegspunkt in das Programm

Variablen



- Zwischenspeichern von Werten in Variablen
  - Datentyp der Variable = primitiv (später Referenzdatentyp)
    - welche Arten von Information können gespeichert werden
  - Bezeichner: Name mit dem Variable identifiziert wird
    - gültiger Name der Variable (später Klasse, Methode, Schnittstelle)
- Beispiele
  - `int anzahl;`
  - `long grosseZahl = 1234567891234567L;`
  - `char eins = '1', zwei = '2', drei = '3';`

Datentypen

- Daten haben unterschiedliche Eigenschaften:
  - `Ganze Zahlen:  2, 4711, -45; ...`
  - `Reelle Zahlen: -3.4, -56.789678, -3.4e-10, 45.567e123, ...`
  - `Buchstaben:    'a', 'b', ...`
  - `Zeichenketten: "hello world!", ...`
- Unterschiedliche Daten benötigen unterschiedlich viel Platz
  - boolesche Werte: 1 Bit
  - Unicode-Zeichen: 16 Bit
  - …
- *Warum braucht man unterschiedliche Datentypen?*

Primitive Datentypen in Java

		Default	Minimum	Maximum
boolean	"1-bit"	`false`	`--`	`--`
byte	8-bit integer	`0`	`-2^7 = -128`	`2^7 -1 = 127`
short	16-bit integer	`0`	`-2^15 = -32768`	`2^15 -1 = 32767`
int	32-bit integer	`0`	`-2^31`	`2^31 -1`
long	64-bit integer	`0L`	`-2^63`	`2^63 -1`
float	32-bit Gleitkommazahl	`0.0F`	`+/-1.4^E-45`	`+/-3.4^E+38`
double	64-bit Gleitkommazahl	`0.0`	`+/-4.9^E-324`	`+/-1.7^E+308`
char	16-bit Unicode	`'\u0000'`	`'\u0000' = 0`	`'\uFFFF' = 65535`

Bauen Sie nie auf den Default-Wert! (Warum?)

Bezeichner

- Identifikation von Variablen (Methoden, Klassen, Schnittstellen)
- Folge von Buchstaben, Zahlen und einigen Sonderzeichen (_, $)
  - beginnt mit _ oder Buchstabe
  - nicht identisch mit Schlüsselwort in Java sein
  - Groß- und Kleinschreibung wird unterschieden

Bezeichner	erlaubt?	Grund
`Mama`	ja	nur Buchstaben
`__Ich_Du2`	ja	Unterstriche und Zahlen erlaubt
`IchBrauche$Geld`	ja	Dollar-Zeichen erlaubtes Sonderzeichen
`2save`	nein	keine Zahl am Anfang erlaubt
`kein Problem`	nein	kein Leerzeichen im Bezeichner erlaubt
`class`	nein	class ist Schlüsselwort in Java

Ausdrücke

- Definition Ausdruck: Verarbeitungsvorschrift, deren Ausführung einen Wert eines bestimmten Typs liefert
  - entsteht, indem Operanden mit Operatoren verknüpft werden
  - Operanden müssen typkonform sein!

```
<Ausdruck> ::=
    <Literal>
  | <Variablenname>
  | "(" <Ausdruck> ")"
  | <unär-Op> <Ausdruck>
  | <Ausdruck> <binär-Op> <Ausdruck>
  | <Ausdruck> <ternär-Op1> <Ausdruck> <ternär-Op2> <Ausdruck>
  | <Funktionsaufruf>
;
```

Literale (Typspezifische Konstanten)

Datentyp	Literal-Definition	Beispiele
boolean	true oder false	`true` `false`

byte, short, int	Zeichenketten aus
	• dezimalen
	• oktalen (führende 0) oder
	• hexadezimalen (führendes 0x)
	Ziffern
		`27`
		
		`035`
		`0x1D`

long	wie bei int mit nachgestelltem L	`27L`
float	wie bei double mit nachgestelltem F	`1.2F`
double	Dezimalzahlen mit Dezimalpunkt und optionalem Exponent	`1.2` `1.2E5  0.23E-3.2`

char	• Zeichen zwischen zwei Apostrophen oder
	• Sonderzeichen mit Escape-Sequenz oder
	• der Unicode-Zeichen
		`'a'  'A'  '2'`
		`'\n'  '\t'`
		`'\u00A9'`

Operatoren

Operator-Typ	Beispiel-Operatoren	Beschreibung
Arithmetik	`+, -, *, /, %`	Mathematik (%=modulo=Rest)
Boolesche Arithmetik (Logik)	`&&, ||, !`	logische Kombination von booleschen Werten; meist zur Definition komplexer Bedingungen
Vergleiche	`==, !=, <, <=, >, >=`	Vergleich von Werten; meist zur Prüfung von Bedingungen
Zuweisung	`=, +=, *=, -=, /=`	Weist Variable einen Wert zu; Kurzformen für relative Wertänderungen (z.B. `+=`)
Inkrement/ Dekrement	`++, --`	spezielle Wertänderungen (+1, -1)
Bit-Operatoren	`<<, >>, &, |, ~, ^,`	Manipulation auf Bit-Ebene; für effiziente komplexe Berechnungen einsetzbar
Spezielle Operatoren	`?: , (type)`	Fallunterscheidung, Typumwandlung

Zahlen Arithmetik

- Arithmetische Operatoren: Ausgabetyp = Eingabetypen
  - Integer-Arithmetik: Typen byte, int, short, long
  - Gleitkomma-Arithmetik: Typen float, double
- Wann sind Rundungsfehler möglich?

Operator	Beschreibung	Integer-Arithmetik	Gleitkomma-Arithmetik
+	positives Vorzeichen	`+234`	`+4.56e7`
-	negatives Vorzeichen	`-456`	`-0.3e-5`
+	Addition	`12+98 (==110)`	`1.2+9.8 (==11.0)`
-	Subtraktion	`12-98 (==-86)`	`1.2-0.9 (==0.3)`
*	Multiplikation	`6*5 = 30`	`6.2*5.7 (==35.34)`
/	Division	`9/4 (==2)`	`9.0/4.0 (==2.25)`
%	Restbildung	`9%4 (==1)`	`9.0%4.1 (==0.8)`

Beispiel

- Summe der ersten n Quadratzahlen

  $1/6n(n + 1)(2n +1)$

- Berechnung für n = 5

Boolesche Arithmetik

- Boolesche Arithmetik
  - Operanden vom Typ boolean
  - gelieferter Wert vom Typ boolean

Operator	Beschreibung	Boolesche Arithmetik
&&	Konjunktion (logisches UND)	`true && false (==false)`
||	Disjunktion (logisches ODER)	`true || false (==true)`
!	Negation (logische Umkehrung)	`!true (==false)`

Vergleiche

- Vergleiche
  - Operanden vom Typ: int, short , long, float, double oder char
  - gelieferter Wert vom Typ boolean

Operator	Beschreibung	Vergleich
==	Gleichheit	`3 == 3 (== true)`
!=	Ungleichheit	`3 != 3 (== false)`
<	Kleiner als	`3.0 < 3.1 (== true)`
<=	Kleiner als oder gleich	`3 <= 3 (== true)`
>	Größer als	`'b' > 'a' (== true)`
>=	Größer als oder gleich	`1 >= -1 (== true)`

Zuweisungen

- Zuweisen eines Werts zu einer Variablen
  - Operanden von beliebigem Typ
  - gelieferter Wert vom Typ der Operanden

  <Zuweisung> ::= <Variablenname> "=" <Ausdruck> ";"
- Zuweisung kann mit Variablendeklaration kombiniert werden
- Zuweisungsoperatoren für relative Änderungen
  - Operator vor = wird auf Variable ausgeführt und Wert der Variable zugewiesen
  - +=, -=, *=, /=
  - superkurz: ++, --

```
int i;          // Deklaration
int k = -34;    // Initialisierung

i = 45;
i = i + 3;
k = i = k+50;

i *= 2;         // i = i*2
k += 100;       // k = k+100
k--;            // k = k-1
```

Bedingungsoperator

- Hilfsoperator für Wertezuweisung inkl. Fallunterscheidung
  - ternärer Operator (= 3 Operanden)
    - erster Operand vom Typ boolean
    - Operanden zwei und drei typkonform
- Syntax: <bedingung> ? <ausdruck1> : <ausdruck2>
  - Semantik: falls <bedingung> true, dann <ausdruck1>, sonst <ausdruck2>

```
x = (i<k)?i-1:j+2
```

identisch zu

```
if (i < k) {
        x = i-1;
} else {
        x = j+2;
}
```

Wann sollten Sie das einsetzen?

Typumwandlungen

- Problem: Sie wollen Variablen mit unterschiedlichen Typen mit Operatoren verknüpfen

```
boolean b = true;
int i = 7;
short s = 9;
char c = 'a';
double d = 7.9;
```

- *Was vom folgenden dürfen Sie tun?*

```
int x = i+s;
short x = i+s;
double x = i+s;
char x = i+s;
int x = b;
int x = c;
```

Implizite Typumwandlung

- Immer wenn vom „kleineren“ auf den „größeren“ Typ ohne Informationsverlust abgebildet werden kann
  - short → int → long
  - char → int
  - float → double
  - natürliche Zahl → Gleitkommazahl

```
int a = 7;
short b = 9;
char c = 'a';
int s1=a+b , s2=a+c, s3=b+c;
double p1 = a*1.0;
```

Explizite Typumwandlungen

- Expliziter Typecast notwendig, wenn Informationsverlust möglich
  - Syntax: (Typ) Ausdruck

```
int    i = 7;
double d1 = (double)i;              // Wert: 7.0

double d2 = 7,9;
int    j = (int)d2;                 // Wert: 7
```

- *Warum macht das Java nicht automatisch, wie z.B. PHP?*
  - Typecast zwischen inkompatible Typen verboten

```
boolean b = true;
int i = (int)b;
```

Präzedenz und Assoziativität

- Präzedenz
  - Punktrechnung vor Strichrechnung
  - Obere Tabelle: Operatoren in absteigender Präzedenz
- Assoziativität
  - Rangfolge bei Operatoren gleicher Präzedenz. untere Tabelle
- Beispiele

  d = 8.2 + 2 * 0.9
  d = 20 / 8 / 2.0
  i = j*= k + 78
  i = j * k / 5

Operatorrangfolge kann durch Klammern überschrieben werden

Operator-Typ	Operatoren (Beispiele)
Präfix/Postfix	`++, --, .`
unäre	`+, -, ~, !`
Erzeugung / Typumwandlung	`new, (type)`
Multiplikation	`*, /, %`
Addition	`+,-`
Vergleich	`==, !=, <, >`
Logisches Und	`&&`
Logisches Oder	`||`
Bedingung	`?:`
Zuweisung	`=, +=, *=`

Operator-Typ	Assoziativität
unär (1 Operand)	rechts
binär (2 Operanden)	links
ternär (3 Operanden)	rechts

Beispiel

- Was ergibt folgender Ausdruck?

```
int i = (int) 5.5f + 4.4f;
```

Zeichenketten

- String: Datentyp für Zeichenketten
  - Verkettung mit „+“
  - Automatische Typumwandlung von allen primitiven Datentypen in Strings

```
String h = "hello";
String w = "world";
String hw = h + " " + w + "!";
// + ist Concatenation-Operator: "hello world!"

int zahl = 4711;
String duft = "koelnisch Wasser = " + zahl;
// implizite Typumwandlung: "koelnisch Wasser = 4711"
```

Lebensdauer von Variablen

- Variablen NUR innerhalb des von „{ }“ umschlossenen Blocks gültig, in oder vor dem sie definiert wurden
  - Ausnahmen: Klassenvariablen und globale Variablen → später

```
double a = 10.3;
{
    double b = 5.5;
    a = a + b;
}
a = a - b;

while (a > 10.0) {
    double c = 3;
    a = a / c;
}
a = a + c;
```

Fehler!
b existiert
nicht mehr

Fehler!
c existiert
nicht mehr

Arrays

Arrays

- **Definition**: Ein Array repräsentiert ein homogenes kartesisches Produkt zur Aufnahme mehrere Daten (Elemente) des gleichen Typs (genannt Elementtyp), wobei auf die Elemente mit Hilfe eines Index zugegriffen wird
- Elementtyp ist fest
- Dimension ist fest
- n-dimensionale Arrays erlaubt
  - n>1 = mehrdimensionales Array
  - Zugriff über n-dimensionalen Index
- random-access-Zugriff (wahlfreier Zugriff)
- Eigenschaften in Java
  - werden dynamisch erzeugt
  - werden von der JVM automatisch gelöscht, wenn sie nicht mehr benutzt werden können

Array Erzeugung

- Array-Erzeugung
  - Anlegen mit dem Schlüsselwort new
  - Anzahl an Elementen der jeweiligen Dimension vorgeben
  - Array-Elemente erhalten Default-Wert des Elementtyps
- Variablen-Zuweisung
  - Gleiche Typen von Array-Variablen und Elementtyp des Arrays(oder Typecast) sowie Arraygrenzen beachten!

```
int[] vektor1;
vektor1 = new int[8]; // reserviert 8 int-Speicherplätze

int i = 3;
float[][] vektor2 = new float[i][2];

char[] alphabet = new double[26];          // Fehler
boolean[] aussagen = new boolean[2][3];    // Fehler
```

Zugriff auf Array Elemente

- Zugriff auf Array-Elemente über einen Index-Wert
  - Index muss zwischen 0 und (Arraydimension -1) liegen
- Initialisierung über Werteliste möglich
  - Arraylänge = Länge der Liste
- Arraylänge über spezielle Variable length

```
int[] vek = new int[5];
vek[0] = -23;
vek[1] = vek[0] + 26;
vek[vek[1]] = -4;
vek[5] = 56; // Laufzeitfehler!
```

```
int[] vek = {-23, 3, 5, -4, 16};
```

```
int l = arr.length;
```

Array Zerstörung

- Java: automatisches Garbage-Collection
  - kein Hantieren mit malloc() oder free() wie in C
  - Speicherplatz wird freigegeben, sobald auf Variable oder Array nicht mehr zugegriffen werden kann

```
int arr[] = {-1, 5, 37};
arr[2] = arr[2] + 1;

arr = new int[15];

{
    double vek[] = new double[10];
    vek[0] = 1.0;
}
```

die alte Belegung von arr ist weg

vek ist weg

Referenzdatentypen

Java ist komisch?

- *Können Sie die folgenden Ausgaben erklären?*

```
String s1 = "Hallo";
String s2 = "Hallo";
System.out.println(s1 == s2);
// Ausgabe: true

int i = 27;
s1 = s1 + i;
s2 = s2 + i;
System.out.println(s1 == s2);
// Ausgabe: false

int arr1[] = {-1, 5, 37};
int arr2[] = {-1, 5, 37};
System.out.println(arr1 == arr2);
// Ausgabe: false
```

Primitive vs Referenzdatentypen

- Variablen mit primitive Datentypen
  - Variablen eines primitiven Datentyps (int, char, double, …)
  - Speichern einen Wert
- Referenzvariablen
  - Referenzdatentyp (Klasse oder Array)
  - Speichern eine **Referenz** auf eine Speicherstruktur, die den Wert enthält *Sie kennen das aus C: Zeiger auf den Speicher*
- Praktische Bedeutung: „==“ vergleicht Referenz, nicht Inhalt!

```
String s1 = "hello" + zahl;
String s2 = "hello" + zahl;
boolean vgl1 = s1 == s2;          // vergleicht Referenz auf String
boolean vgl2 = s1.equals(s2);     // vergleicht Inhalt vom String
```

Referenzdatentypen

- Arrays und Klassen sind Referenzdatentypen
  - Array- und Objektvariablen speichern Referenzen

- Referenzen ungleich Speicheradressen!
  - Speichermanagement übernimmt die JVM
  - JVM kann Speicher verschieben oder freigeben „Garbage Collector“
    - vgl. Arrays
    - keine Zeigerarithmetik wie in C, C++

Was ist also passiert?

```
String s1 = "Hallo";
String s2 = "Hallo";
System.out.println(s1 == s2);
// Ausgabe: true
```

Optimierer des Java-Compilers hat s1 und s2 auf die selbe Referenz abgebildet
→ Vergleich ergibt true

```
int i = 27;
s1 = s1 + i;
s2 = s2 + i;
System.out.println(s1 == s2);
// Ausgabe: false
```

s1 und s2 wurden geändert, neue Referenzen
→ Vergleich ergibt false

```
int arr1[] = {-1, 5, 37};
int arr2[] = {-1, 5, 37};
System.out.println(arr1 == arr2);
// Ausgabe: false
```

Optimierer hat's offenbar nicht gemerkt ;-)
→ Vergleich ergibt false

Was bedeutet das in der Praxis?

- Unterschiedliche Handhabung von primitiven Datentypen und Referenzdatentypen bei
  - Literale
  - Zuweisung und Gleichheit
  - Parameterübergabe
  - Funktionswerte

Literale

- Literal für Referenzdatentypen: null
  - Entspricht: Referenzvariable ist kein Wert zugeordnet

```
int[] arr = null;
System.out.println(arr);
// Ausgabe: null

arr = new int[3];
arr[0] = 4;
System.out.println(arr);
// Ausgabe: Referenz, z.B. [I@2a139a55

arr = null;
// Speicherbereich freigegeben, da nicht mehr referenziert
```

Literale

- String ist ebenfalls Referenzdatentyp, also

```
String s = "abc"; // ist Kurzform für
String s = new String("abc");
String s = null; // kein String vorhanden
```

Zuweisungsoperator

- Zuweisung weist Referenz zu, nicht Werte!
- a=b bedeutet: Arrayvariable a verweist auf die gleiche Referenz wie Arrayvariable b

```
int[] a = {0,2};
int[] b = {1,3};

a=b;

a[1] = 31;
```

```
int[] a = {0,2};
int[] b = {1,3};

a[0]=b[0];
a[1]=b[1];

a[1] = 31;
```

- *Welcher Wert steht am Ende in b[1]?*

Prüfung auf Gleichheit

- Gegeben zwei Referenz-Datentypen a und b
- **Wertegleichheit**
  - a und b haben den selben Inhalt
  - Für Referenzdatentypen existiert eine equals()-Methode
- **Referenzgleichheit**
  - a und b verweisen auf die selbe Referenz(d.h., Speicherstelle)
  - Prüfung mit „==“
- *Achtung: equals() liefert bei Arrays dasselbe wie „==“*

```
int i = 27;
String a = "Hallo"+i;
String b = "Hallo"+i;

System.out.println(a == b);
// Ausgabe: false

System.out.println(a.equals(b));
// Ausgabe: true

a = b;
System.out.println(a == b);
// Ausgabe: true

System.out.println(a.equals(b));
// Ausgabe: true
```

Parameterübergabe bei Methodenaufruf

- Werden einer Funktion Referenzdatentypen übergeben, haben Änderungen an den Daten Auswirkungen über die Methodengrenzen hinaus!

```
static void test(int[] i) {
    i[1] = 38;
}

int i[] = {1,2,3,4,5};
test(i);
System.out.println(i[1]);
```

- *Was ist die Ausgabe? Wann sollten Sie so etwas nicht tun?*

- *Was ändert sich, wenn die Methode aussieht wie folgt?*

```
static void test(int[] i) {
    i = new int[5];
    i[1] = 38;
}
```

Ein – und Ausgabeanweisungen

- Ausgabe (auf Standard-Output, d.h. in der Regel Bildschirmconsole)
  - System.out.print(<Ausdruck>); // ohne Zeilenumbruch
  - System.out.println(<Ausdruck>); // mit Zeilenumbruch

- Eingabe (auf Standard-Input, d.h. in der Regel Tastatur in Console)
  - Eingabeverarbeitung in Java als Stream (Folge von Zeichen)
  - Mögliches Vorgehen:
    - Definition eines Scanners für Standard-Input
    - nextLine() liefert mit Enter abgeschlossene Eingaben

- Konvertierung von Texteingaben (String) in Zahlen
  - Integer.parseInt(<String-Ausdruck>); ?? Can‘t we parse int right away?

Beispiel Ein – und Ausgabe

```
import java.util.Scanner;

public class EingabeAusgabe {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    // Ausgabe
    System.out.print("Hallo "); // ohne Zeilenumbruch
    System.out.println("Welt!"); // mit Zeilenumbruch (ln)
    // Eingabe Strings
    String ersteZeile = sc.nextLine();
    String zweiteZeile = sc.nextLine();
    System.out.println("Erste Zeile: " + ersteZeile);
    System.out.println("Zweite Zeile: " + zweiteZeile);
    // Eingabe Zahlen
    int zahl = sc.nextInt();
    zahl = zahl * zahl;
    System.out.println("Quadrat der eingegebenen Zahl: " + zahl);
  }
}
```

Zusammenfassung

- Imperative Programmierung
  - sequentielle Verarbeitung von Daten/Werten
  - Deklarationen, Zuweisungen, Anweisungssequenzen
- Datentypen
  - Zusammenfassung von Wertebereichen und Operationen
- Variablen
  - logische Behälter zur Speicherung von Werten
- Ausdruck
  - Verarbeitungsvorschrift, deren Ausführung einen Wert liefert
- Operatoren
  - Verknüpfung von Ausdrücken

Kontrollfragen

- Wie sieht das Grundgerüst eines Java-Programms aus?
- Grenzen Sie die implizite und die explizite Typumwandlung voneinander ab.
- Welche Merkmale haben Arrays, wie werden sie angelegt und wie wird auf sie zugegriffen?
- Was sind die Unterschiede bei der Deklaration und Verwendung von primitiven Datentypen und Referenzdatentypen?
- Erläutern Sie die Sichtbarkeit und Lebensdauer von Variablen.