1 Imperative Programmierelemente

Einlesen von der Tastatur

Um einen Text von der Tastatur einzulesen gibt es das Objekt `System.in`. Ohne nähere Erklärung der Hintergründe soll hier zunächst vorgestellt werden, wie man damit eine Ganzzahl von der Tastatur einlesen kann.

```
Scanner eingabe = new Scanner(System.in);
```

Damit hat man ein Objekt erzeugt, das eine Methode zum Einlesen einer Zeichenkette hat und kann nun über die entsprechende Methode zeilenweise von der Tastatur lesen.

Beispiel: Einlesen von Zahlen

```
import java.until.*;
public class Einlesen {
  public static void main(String[] args) {
    Scanner eingabe = new Scanner(System.in);

    int a, b;
    System.out.println("Bitte Zahl a eingeben: ");
    a = eingabe.nextInt();
    System.out.println("Bitte Zahl b eingeben: ");
    b = eingabe.nextInt();

    System.out.println("Sie haben die Zahlen a =" + a + " und b = " + b + " eingegeben.");
  }
}
```

Mit dem `Scanner` können die unterschiedlichsten Datentypen eingelesen werden.

Methode	**Ergebnis der Methode**
`next()`	String (trennt nach Leerzeichen)
`nextLine()`	String (speichert gesamte Eingabe in einer Variablen)
`nextBoolean()`	erzeugt aus der Eingabe ein `boolean` und gibt diesen zurück
`nextDouble()`	Konvertiert die Eingabe zum Typ `double`
`nextFloat()`	Gibt die Eingabe als Typ `float` zurück
`nextByte()`	Die Eingabe wird als `byte` zurückgegeben
`nextShort()`	Konvertiert die Eingabe zum Typ `short`
`nextInt()`	Die Nutzereingabe wird als `int` zurückgegeben
`nextLong()`	Erzeugt aus der Eingabe einen Wert vom Typ `long`

1.1 Datentypen

Java kennt folgende Grunddatentypen:

- ganzzahlige Datentypen (Integer-Datentypen): `byte`, `short`, `int`, `long`
- Fließkomma-Datentypen (Gleitkomma-, Floating-Point-Datentypen): `float`, `double`
- Zeichentyp: `char`
- logischer Datentyp (Boolescher Datentyp): `boolean`

Die Wertebereiche der Datentypen sind in Java unabhängig vom Rechnersystem und der Java-Implementierung.

Außer der nachfolgend beschriebenen Grunddatentypen kennt Java noch die so genannten Referenzdatentypen, in denen Referenzen auf Objekte gespeichert werden können. Ein solcher Referenzdatentyp ist `String`.
Prinzipiell gilt: Alle nicht-Grunddatentypen sind Referenzdatentypen.

1.1.1 Ganzzahliger Datentyp

Ganzzahlige Datentypen enthalten ganze Zahlen (ganzzahlige Werte) mit Vorzeichen.

Typname	Länge	Wertebereich
byte	8 Bit	$-128 \ldots 127$
short	16 Bit	$-32768 \ldots 32767$
int	32 Bit	$-2^{31} \ldots 2^{31}-1$
long	64 Bit	$-2^{63} \ldots 2^{63}-1$

Beispiel:

```
int zahl1;          //Datentyp int, Name zahl1
zahl1 = 1099;       //Wert 1099 wird zugewiesen.
```

Zulässige Werte für `zahl1` sind $-1000, 999, +2001, \ldots$.
Ganzzahlige Literalte sind grundsätzlich vom Typ `int`.

1.1.2 Fließkommazahlen

Fließkommazahlen sind immer Dezimalzahlen. Sie bestehen aus Vorzeichen, Vorkommateil, Dezimalpunkt, Nachkommateil, Exponent, Suffix (f,F, d, D).
Fließkomma-Literale sind in Java standardmäßig vom Typ `double`.

Typname	Länge	Wertebereich
float	32 Bit	$+/- 3,40282347 * 10^{38}$
double	64 Bit	$+/- 1,79769313486231570 * 10^{308}$

Beispiel:

```
float fa;
double da, db;
fa = 1.1f;
da = 22.0;
db = 1.02*3.3E27;
```

1.1.3 Zeichentyp (Character-Datentyp)

Der Character-Datentyp ist ein Datentyp zur Darstellung von einzelnen Zeichen. Der Typ `char` hat in Java einen Wertebereich von 16 bit. Der Zeichensatz in Java ist der Unicode-Zeichensatz. Character-Zeichen werden in einfachen Hochkommata eingeschlossen.

Beispiel:

```
char ca, coe, cnl, ccr, chk, ct;
ca = 'A'; coe='Ö';          //Ausgabe hängt vom Ausgabesystem ab
cnl = '\n';                 //Escape-Sequenz[1] für einen Zeilenumbruch (new line)
ccr = '\r';                 //Escape-Sequenz[1] carriage return (zurück zum Zeilenanfang)
chk = '\'';                 //Darstellung Hochkomma
ct = '\t';                  //Tabulator
```

Java kennt die "üblichen" Escape-Sequenzen[1] zur Darstellung von Sonderzeichen und Formatierung:

Zeichen	**Bedeutung**
\t	Horizontaler Tabulator
\n	Zeilenumbruch
\f	Seitenumbruch
\"	Doppeltes Anführungszeichen
\'	Einfaches Anführungszeichen
\\	Backslash

1.1.4 Boolescher Datentyp (boolean)

Java kennt den Datentyp `boolean` zur Darstellung logischer Werte (wahr oder falsch). Variablen vom Typ `boolean` können somit nur zwei Werte annehmen (auf Syntaxebene dargestellt durch `true` und `false`). Intern ist die Länge von `boolean` immer 1 Byte.

Beispiel:

```
boolean janein;
janein = true;
```

Java erlaubt keine automatische Umwandlung von Integer-Werten in boolesche Werte (wie dies in C möglich ist). Deshalb dürfen ganzzahlige Typen mit den Werten 0 oder 1 nicht als Ersatz für die logischen Typen verwendet werden.

Konvertierung von Grunddatentypen

Java führt automatisch nur Typkonvertierungen vom "kleineren" zum "größeren" Typ durch. Automatisch durchgeführt werden folgende Konvertierungen:
`byte` → `short` → `int` → `long` → `float` → `double` und `char` → `int`
Konvertierungen in die andere Richtung erfordern eine explizite Angabe des Zieldatentyps in runden Klammern (**Cast-Operator**). Der Ausdruck (`type`)`a` wandelt den Ausdruck `a` in einen Ausdruck vom Typ `type` um.

Mit Hilfe des Type-Cast-Operators dürfen alle legalen Typkonvertierungen vorgenommen werden. Der Type-Cast-Operator wird vor allem dann angewendet, wenn der Compiler keine implizite Konvertierung vornimmt; beispielsweise bei der Zuweisung von größeren an kleinere numerische Typen oder bei der Umwandlung von Objekttypen.

Beispiel: Typkonvertierung

```
char c;
int i,j;
float x,y;
double v,w;
i = 3; c = 'a';
j = c;          //Implizite Konvertierung von char nach int
x = i*j;        //Implizite Konvertierung von int nach float
v = 3.0; w = 2.0;
i = (int)(v/w);//Konvertierung mit cast-Operator von double nach int.
                //ergibt die Ziffern vor dem Komma
x = 99.0f       //Das Literal f gibt an, dass es sich um eine Zahl vom Typ float handelt.
y = 100.01f;
v = x*y;        //Implizite Konvertierung von float nach double
x = (float)v;//Konvertierung von double nach float
```

1.2 Strings

Strings (Zeichenfolge, Zeichenkette) sind Folgen von Einzelzeichen. String-Literale werden in Java durch Hochkommata eingeschlossen. Sie sind in Java Objekte der API-Klasse `String`.

Beispiel:

```
String meinString;
meinString = "Hallo Java kurs."
```

Der Operatr `new` darf bei der Instanzierung von Strings mittels Literalen entfallen. Die Einzelzeichen von Strings sind über einen Index ansprechbar. Die Index-Zählung beginnt bei Null.

Beispiel:
Der String "Beispiel" hat eine Länge von 8 Zeichen.
Das erste Zeichen an der Position 0 ist 'B', das letzte Zeichen an der Position 7 ist 'l'.

Ein String kann auch nicht druckbare Zeichen enthalten wie z.B. `\n`= Neue Zeile.

1.2.1 String-Verkettung und Konvertierung zu String

Strings können durch den Operator "+" miteinander verbunden werden.

```
String s1 = "Hallo";
String s2, s3, s4;
s2 = " und ";
s3 = "Java";
s4 = s1 + s2 + s3;
```

Daten werden durch den String-Operator "+" automatisch in den String-Typen umgewandelt.

```
int i = 999;
String ausgabe = "i ist " + i;
```

1.2.2 String-Bearbeitung

Möchte man die gespeicherte Zeichenkette bearbeiten bzw. auf einen bestimmten Buchstaben an einem Index zugreifen, so bietet Java hier diverse Möglichkeiten. Die API-Klasse `String` stellt mehr als 50 Methoden zur Bearbeitung von Strings zur Verfügung.

Beispiel:

```
String s = "Dies ist ein String mit 35 Zeichen."
```

Zugriff auf ein Zeichen

Mit `s.charAt(i)` kann das Zeichen (Typ `char`) des Strings `s` an der i-ten Poistion bestimmt werden. Dabei läuft der Index `i` wie bei allen Indizes von 0 bis `s.lenth()-1`

Zugriff auf eine Zeichenkette innerhalb eines Strings

Die Methode `s.subString(Startindex, Endindex)` liefert die Teilzeichenkette (Typ `String`) beginnend bei der *Startposition* bis hin zur *Endposition* zurück.

Länge der Zeichenkette

`s.length()`liefert die aktuelle Länge des String-Objekts `s`. Wenn der String leer ist, ist der Rückgabe-

wert von `length()` geich 0. Wird ein Wert *n* größer 0 zurückgegeben, so enthält der String *n* Zeichen, die an den Indexpositionen 0 bis *n-1* liegen.
Für das obige Beispiel liefert `s.length()` also den Wert 35.

Vergleich von Zeichenketten

Da Strings Referenzdatentypen sind, kann man zwei Strings nicht mittels `if (S1 == s2)...` auf inhaltliche Gleichheit prüfen! Der Code `if (s1 == s2)...` würde bedeuten, dass überprüft wird, ob beide Stringvariablen auf den selben String weisen.

- `if(s1.equals(s2))...` bedeutet, falls die Strings `s1` und `s2` gleich sind, ...
- Bei `s1.equalsIgnoreCase(s2)` wird die Groß- und Kleinschreibung nicht unterschieden.
- Der Aufruf von `s1.compareTo(s2)` gibt die Ganzzahl 0 zurück, falls die Zeichenketten gleich sind, einen negativen Wert, wenn `s1` lexikographisch (also im Alphabet) vor `s2` steht und sonst einen positiven Wert.

Weitere String-Methoden:

- `String substring(int start, int ende);` liefert den Teilstring, der an der Position `start` beginnt und an der Position `end-1` endet. Wie bei allen Zugriffen über einen numerischen Index beginnt auch hier die Zählung bei 0.
- `s.toLowerCase()` wandelt den String in Kleinbuchstaben um
- `s.toUpperCase()` wandelt in Großbuchstaben um
- `s.trim()` entfernt Leerzeichen am Anfang und am Ende des Strings
- `s.replace(String alt, String neu)` ersetzt die Zeichenkette `alt` durch die Zeichenkette `neu`

Beispielprogramm zum Umgang mit Strings:

```
public class StringBeispiele {
  public static void main(String[] args) {
    String s1 = "Zeichenkette";
    String s2, s3, s4;

    char first = s1.charAt(0);
    char last = s1.charAt(s1.length()-1);
    System.out.println("\n erstes Zeichen: " + first +
      "\n\n letztes Zeichen: " + last);

    s2 = s.toUpperCase();
    System.out.println("\n " + s2);

    String s5 = "n bearbeiten";
    s3 = s1 + s5;
    System.out.println("\n Stringverkettung: " + s3);
```

```
        int laenge = s3.length();
        System.out.println("\n Der String hat " + laenge + " Zeichen.");

        s4 = s3.substring(7,11);
        System.out.println("\n Teilstring von Zeichen 7 bis 10: " + s4);
      }
    }
```

Ausgabe:
erstes Zeichen: Z
letztes Zeichen: e
ZEICHENKETTE
Stringverkettung: Zeichenketten bearbeiten
Der String hat 24 Zeichen.
Teilstring von Zeichen 7 bis 10: kett

Methoden der Klasse String

- `s.length()` liefert die aktuelle Länge des String-Objekts s
- `s.substring(int start, int end)` liefert den Teilstring, der an der Position `start` beginnt und an der Position `end-1` endet.
- `str1.equals(String str2)` prüft die Strings `str1` und `str2` auf Gleichheit
- `s.toLowerCase()` wandelt den String in Kleinbuchstaben um
- `s.toUpperCase()` wandelt den String in Großbuchstaben um
- `s.trim()` entfernt die Leerzeichen am Anfang und am Ende
- `s.indexOf(String str)` sucht das erste Vorkommen der Zeichenkette `str` innerhalb des String.-Objekts `s`. Wird `str` gefunden, liefert die Methode den Indes des ersten übereinstimmenden Zeichens zurück. Ansonsten wird -1 zurückgegeben.
- `s.indexOf(String str, int fromIndex)` erfüllt die gleiche Funktionalität wie zuvor genannte Methode. Die Suche beginnt aber erst ab der durch `fromIndex` gegebenen Position. Wird `str` beginnend ab dieser Position gefunden, liefert die Methode den Index des ersten übereinstimmenden Zeichens, andernfalls -1.
- `s.lastIndexOf(String str)` sucht nach dem letzen Vorkommen des Teilstrings `str` um aktuellen String-Objekt. Wird `str` gefunden, liefert die Methode den Index des ersten übereinstimmenden Zeichens, andernfalls -1.
- `s.replace(char altesZeichen, char neuesZeichen)` führt eine zeichenweise Konvertierung des aktuellen String-Objekts durch. Dabei wird jedes Vorkommen der Zeichenkette `altesZeichen` durch die neue Zeichenkette `neuesZeichen` ersetzt.

- `s.replace(String alt, String neu)` führt eine zeichenweise Konvertierung des aktuellen String-Objekts durch. Dabei wird jedes Vorkommen der Zeichenkette `alt` durch die neue Zeichenkette `neu` ersetzt.

- `s.compareTo(String str)` vergleicht die Strings `s` und `str` lexikographisch. Das heißt, die Strings werden paarweise von inks nach rechts verglichen. Tritt eine Unterschied auf, oder ist einer der Strings beendet, wird das Ergebnis ermittelt.
  Ost das aktuelle String-Objekt `s` dabei kleiner als `str`, wird ein negativer Wert zurückgegeben. Ist es größer, wird ein positiver Wert zurückgegeben..
  Bei Gleichheit liefert die Methode den Rückgabewert 0.

1.3 Arrays

1.3.1 Eigenschaften

- Liste von Daten gleichen Typs
- Zugriff auf Elemente der Liste erfolgt über Laufindizes (Laufvariablen).
- Laufindizes werden in eckigen Klammern [] gesetzt und sind ganzzahlig
- Wert der Laufindizes beginnt mit 0
- Jedes Array verfügt über die Instanzvariable `length`, welche die Anzahl der Elemente enthält.
- Die Anzahl der Array-Elemente wird bei Instanzierung des Arrays festgelegt und ist nicht dynamisch veränderbar.
- Arrays sind Objekte!

Beispiel:
Es sollen mehrere Namen als String gespeichert werden. Anstatt für jeden Namen eine Stringvariable anzulegen (etwa `name1`, `name2`, ...) wird ein Array von Strings erzeugt.

```
public class StringArray {
  public static void main(String[] args) {
    String[] name = new String[3];
    name[0] = "PKW";
    name[1] = "Bus";
    name[3] = "Zug";
    while (i < name.length) {
      System.out.println(name[i]);
      i++;
    }
  }
}
```

Ausgabe:
PKW

Bus
Zug

1.3.2 Deklaration von Arrays

Die Deklaration eines Arrays in Java erfolgt in zwei Schritten:

1. Deklaration einer Array-Variablen
   Beispiel:
   ```
   int[] a;
   double b[];
   boolean [] c;
   ```
   Die Deklaration eines Arrays entspricht der einer einfachen Variable, mit dem Unterschied, dass an den Typ oder Variablennamen eckige Klammern angehängt werden. Die eckigen Klammern können auch vor dem Variablennamen stehen.

2. Erzeugen eines Arrays und Zuweisung an die Array-Variable
   Zum Zeitpunkt der Deklaration wird noch nicht festgelegt, wie viele Elemente das Array haben soll. Dies geschieht bei der Initialisierung mit Hilfe des `new`-Operators.
   Beispiel:
   ```
   a = new int[6];
   b = new double[10];
   c = new boolean[15];
   ```

Ist bereits zum Deklarationszeitpunkt klar, wie viele Elemente das Array haben soll, können Deklaration und Initialisierung zusammengeschrieben werden:

```
int[] a = new int[6];
```

Wertzuweisung in Arrays
Die Wertzuweisung erfolgt durch den Zugriff auf das Array-Element unter Angabe des entsprechenden Index.
Beispiel:

```
b[0] = 0.99;    //Zugriff auf das erste Element
b[4] = 0.001;   //Zugriff auf das 5. Element
```

1.3.3 Literale Initialisierung von Arrays

Alternativ zur Verwendung des `new`-Operators kann ein Array auch literal initialisiert werden. Dazu werden die Elemente des Arrays in geschweifte Klammern gesetzt. Die Größe des Arrays ergibt sich aus der Anzahl der zugewiesenen Elemente.

Beipiel:

```
int[] x = {1, 2, 6, 4, 8}    //int-Array mit 5 Elementen
boolean[] y = {true, true}   //boolean-Array mit 2 Elementen
```

Die literalte Initialisierung muss unmittelbar bei der Deklaration erfolgen.

Programmbeispiel Arrays:

```
public class Array1 {
  public static void main(String[] args) {
    int[] a = new int[3];
    boolean[] b = {true, false};
    a[0] = 2;
    a[1] = 2;
    a[2] = 5;

    System.out.println("a hat " + a.length + " Elemente.");
    System.out.println("b hat " + b.length + " Elemente.");
    System.out.println(a[0]);
    System.out.println(a[1]);
    System.out.println(a[2]);
    System.out.println(b[0]);
    System.out.println(b[1]);
  }
}
```

1.3.4 Mehrdimensionale Arrays

Mehrdimensionale Arrays werden erzeugt, indem zwei oder mehr Paare eckiger Klammern bei der Deklaration angegeben werden.

```
int[][] c = new int [2][7];
```

c ist ein Array mit $\underbrace{2}_{1.\ Klammerausdruck}$ Spalten und $\underbrace{7}_{2.\ Klammerausdruck}$ Zeilen.

Die Initialisierung erfolgt wie bei den eindimensionalen Arrays durch Angabe der Anzahl der Elemente. Der Zugriff auf mehrdimensionale Arrays erfolgt duch die Angabe aller erforderlichen Indizes, jeweils durch eigene eckige Klammern.

Beispiel:

```
c [1] [1] = 4;    //In der zweiten Spalte, zweite Zeile wird der Wert 4 gespeichert.
c [1] [2] = 6;    //In der zweiten Spalte, dritte Zeile wird der Wert 6 gespeichert.
```

Programmbeispiel mehrdimensionale Arrays:

```
public class Array2 {
  public static void main(String[] args) {
```

```
        int [][] a = new int[2][3];
        a[0][0] = 1;  a[1][0] = 4;
        a[0][1] = 2; a[1][1] = 5;
        a[0][2] = 3; a[1][2] = 6;

        for(int i = 0; i < = 2; i++) {
          System.out.println(" " + a[0][i] + " " + a[1][i]);
        }
      }
    }
```

Ausgabe:

1 4
2 5
3 6

1.4 Operatoren

Java kennt die "üblichen" Operatoren, dazu gehören:

- arithmetische Operatoren
- Zuweisungsoperatoren
- Vergleichsoperatoren
- logische Operatoren

Weitere Operatoren sind Bitoperatoren, Konditionaloperatoren und Stringverkettungen.

1.4.1 Arithmetische Operatoren

	Bezeichnung	**Bedeutung**
$+$	Positives Vorzeichen	$+n$ ist gleichbedeutend mit n
$-$	Negatives Vorzeichen	$-n$ kehrt das Vorzeichen von n um
$+$	Summe	$a + b$ ergibt die Summe von a und b
$-$	Differenz	$a - b$ ergibt die Differenz von a und b
$*$	Produkt	$a * b$ ergibt das Produkt aus a und b
$/$	Quotient	a/b ergibt den Quotienten von a und b
$\%$	Restwert	$a\%b$ ergibt den Rest der ganzzahligen Division von a durch b. In Java lässt sich dieser Operator auch auf Fließkommazahlen anwenden.
$++$	Präinkrement Postinkrement	`++a` erhöht a um 1 und gibt dann das Ergebnis zurück `a++` gibt zunächst a zurück und erhöht dann a um 1
$--$	Prädekrement Postdrekement	`--a` verringert a um 1 und gibt dann das Ergebnis zurück `a--` gibt zuerst a zurück und verringert dann a um 1

1.4.2 Zuweisungsoperatoren

	Bezeichnung	**Bedeutung**
$=$	Einfache Zuweisung	`a = b` weist a den Wert von b zu und liefert b als Rückgabewert.
$+=$	Additionszuweisung	`a += b` weist a den Wert von $a+b$ zu und liefert $a+b$ als Rpckgabewert.
$-=$	Subtraktionszuweisung	`a -= b` weist a den Wert von $a-b$ zu und liefert $a-b$ als Rückgabewert.
$*=$	Multiplikationszuweisung	`a *= b` weist a den Wert von $a*b$ zu und liefert $a*b$ als Rückgabewert.
$/=$	Divisionszuweisung	`a /= b` weist a den Wert von a/b zu und liefert a/b als Rückgabewert.
$\%=$	Modulozuweisung	`a %= b` weist a den Wert von $a\%b$ zu und liefert $a\%b$ als Rückgabewert.
$\&=$	UND-Zuweisung	`a &= b` weist a den Wert von $a\&b$ zu und liefert $a\&b$ als Rückgabewert.
$\mid=$	ODER-Zuweisung	`a= b|` weist a den Wert von $a\|b$ zu und liefert $a\|b$ als Rückgabewert.
$\hat{}=$	Exklusiv-ODER-Zuweisung	`a ^= b` weist a den Wert von a^b zu und liefert a^b als Rückgabewert.

1.4.3 Relationale Operationen

Relationale Operatoren vergleichen Ausdrücke und liefern einen logischen Rückgabewert (wahr oder falsch/`true` und `false`).

Operator	**Bezeichnung**	**Bedeutung**
$==$	Gleichheit	`a == b` ergibt `true`, wenn a gleich b ist. Sind a und b Referenztypen, so ist der Rückgabewert `true`, wenn beide Werte auf dasselbe Objekt zeigen.
$!=$	Ungleich	`a != b` ergibt `true`, wenn a ungleich b ist. Sind a und b Objekte, so ist der Rückgabewert `true`, wenn beide Werte auf unterschiedliche Objekte zeigen.
$<$	Kleiner	`a < b` ergibt `true`, wenn a kleiner b ist.
$<=$	Kleiner gleich	`a <= b` ergibt `true`, wenn a kleiner oder gleich b ist.
$>$	Größer	`a > b` ergibt `true`, wenn a größer als b ist.
$>=$	Größer gleich	`a >= b` ergibt `true`, wenn a größer oder gleich b ist.

1.4.4 Logische Operatoren

Logische Operatoren verknüpfen boolesche Werte, liefern als Rückgabewert ebenfalls boolesche Werte.

Operator	Bezeichnung	Bedeutung
!	logisches NICHT	`!a` ergibt `false`, wenn a wahr ist. Der Ausdruck ergibt `true`, wenn a falsch ist.
&&	UND mit ShortCircuit-Evaluation	`a&&b` ergibt `true`, wenn sowohl a als auch b wahr sind. Ist bereits a falsch, so wird direkt `false` zurückgegeben und b wird nicht mehr ausgewertet.
||	ODER mit Short-Circuit-Evaluation	`a|b` ergibt `true`, wenn mindestens einer der beiden Ausdrücke a oder b wahr ist. Ist bereits a wahr, so wird direkt `true` zurückgegeben und b wird nicht mehr ausgewertet.
&	UND ohne Short-Circuit-Evaluation	`a & b` ergibt `true`, wenn sowohl a als auch b wahr sind. Beide Teilausdrücke werden unabhängig ihrer Ergebnisse ausgewertet.
|	ODER ohne Short-Circuit-Evaluation	`a|b` ergibt `true`, wenn mindest einer der beiden Ausdrücke a oder b wahr ist. Beide Teilausdrücke werden unabhängig ihrer Ergebnisse ausgewertet.
^	Exklusiv-ODER	`a^b` ergibt `true`, wenn beide Ausdrücke einen unterschiedlichen Wahrheitswert haben.

1.5 Kontrollstrukturen

Java kennt folgende Kontrollstrukturen:

- Entscheidungsanweisungen `if`, `if-else`
- Zählschleife `for`
- Abweisende Schleife `while`
- Nicht-abweisende Schleife `do-while`
- Entscheidungsanweisungen `switch`

1.5.1 Entscheidungsanweisung if-else

Die Syntax:

```
if (bedingugn) {
  anweisungen1;
} else {
  anweisungen2;
}
```

Die Bedeutung und Wirkung:

Die Bedingung `bedingung` wird ausgewertet. Falls die Bedungung wahr (`true`) ist, wird der Anweisungsblock `anweisungen1` (oder die Anweisung) ausgeführt.
Ist die Bedingung nicht wahr (`false`) wird der `else`-Zwei ausgeführt.

Beispiel:

```
public class ScheifenIf {
  public static void main(String[] args) {
    int zahl;
    zahl = 2*3;
    if (zahl == 6) {
      System.out.println("Das Ergebnis ist sechs.");
    } else {
      System.out.println("Das Ergebnis ist nicht sechs.");
    }
  }
}
```

1.5.2 Zählschleife for

Syntax:

```
for (initialisierung; testbedingung; zaehlanweisung) {
  anweisungen;
}
```

Beispiel:

Die Zahlen 1 bis 10 sollen ausgegeben werden:

```
for(int i = 1; i <= 10; i++) {
  System.out.println(i);
}
```

Erläuterung des Schleifenkopfes:

Fange an mit `i = 1`. Prüfe, ob i noch kleiner oder gleich 10 ist, wenn ja, durchlaufe die Schleife und erhöhe i um eins, u.s.w.

`initialisierung` Im Beispiel: `int i = 1;`	Start (Initialisierungs-) Anweisung wird vor dem Ausführen der Schleife **einmal** ausgeführt. Deklarationen von Variablen sind zulässig, die Variable ist nur innerhalb der Schleife sichtbar. Im Beispiel: Variable i wird als Zählindex vereinbart.

`testbedingung` Im Beispiel: `i <= 10;`	Testbedingung für den Abbruch der Schleife. Die Schleife wird solange ausgeführt, wie die Testbedingung wahr ist (den Rückgabewert `true` liefert). Die Testbedingung wird **vor jedem** Schleifendurchlauf abgeprüft. Im Beispiel: Die Schleife wird ausgeführt solange `i` kleiner oder gleich 10 ist.
`zaehlanweisung` Im Beispiel: `i++`	In der Zehlanweisung wird in der Regel der Schleifenzähler verändert. Der Ausdruck wird **nach jedem** Durchlauf der Schleife ausgewertet. Im Beispiel: Zählindex i wird mit dem Inkrementoperator bei jedem Schleifendurchlauf um 1 erhöht.

Jeder der drei Schleifenkopf-Ausdrücke darf auch fehlen. Fehlt die Testbedingung, so setzt der Compiler automatisch die Konstante `true` ein. Initialisierung und Zählanweisung dürfen auch aus mehreren Ausdrücken bestehen, die mit Komma getrennt werden.

1.5.3 Abweisende Schleife while

Syntax:

```
while (bedingung) {
        anweisungen;
}
```

Die Anweisungen im Schleifenrumpf werden solange ausgeführt, wie die Bedingung im Schleifenkopf wahr ist. Ist die Bedingung bereits vor dem ersten Eintritt in den Schleifenrumpf falsch,wird der Schleifenrumpf nicht durchlaufen.

Beispiel:

```
int j = 0;
while (j < 10) {
  System.out.println(j++);
}
```

Die Schleife wird solange durchlaufen, wiew j kleiner als 10 ist und gibt die Zahlen 0 bis 9 aus.

1.5.4 Nicht-abweisende Schleife do-while

Syntax:

```
do {
  anweisungen;
} while (bedingung);
```

Die Anweisungen im Schleifenrumpf werden solange ausgeführt, wie die Bedingung wahr ist. Die Anweisungen werden mindestens einmal ausgeführt.

Beispiel:

```
int j = 0;
do {
  System.out.println(j++);
} while (j < 10);
```

1.5.5 `break` und `continue` in Schleifen

Die normale Schleifenreihenfolge von for, while und do-while Schleifen kann mittels der beiden Anweisungen `break` und `continue` beeinflusst werden:

- `break` verlässt die Schleife und fährt mit der Ausführung der ersten Anweisung hinter der Schleife fort
- `continue` springt an das Ende des Schleifenrumpfes und beginnt sofort den nächsten Schleifendurchlauf

1.5.6 Merfachverzweigung switch

Syntax:

```
switch (wert) {
  case const1: anweisung1; break;
  case const2: anweisung2; breack;
  ...
  default: default_anweisung;
}
```

Der `switch`-Anweisung erwartet einen Wert (Ausdruck mit Rückgabewert) vom Typ `int`, `char`, `byte` oder `short`. Es können beliebig viele Verzweigungen mit `case` folgen.
Zu jeder `case`-Verzweigung gehört eine Konstante (`const1`, `const2`,...).
Stimmt der Wert einer Konstante in einer `case`-Verzweigung mit dem aktuellen Wert im `switch`-Ausdruck überein, werden alle Anweisungen ab dem entsprechenden `case`-Zweig ausgeführt. `break` verlässt die `switch`-Anweisung.

Achtung: `case` ist ein Einsprungsziel! Fehlt `break` in einem `case`-Zweig, werden die folgenden `case`-Zweige ebenfalls durchlaufen.

Beispiel:

```
int zahl;
zahl = (int)(Math.random()*4);
switch (zahl)
{
  case 1 : System.out.println(zahl + " Eins "); break;
  case 2 : System.out.println(zahl + " Zwei "); break;
  case 3 : System.out.println(zahl + " Drei "); break;
  default: System.out.println(zahl + " Kein Treffer ");
}
```

Die `int`-Variable `zahl` erhält einen Zufallswert 0, 1, 2 oder 3. Je nach Wert von `zahl` wird der `case`-Block 1, 2, 3 oder `default` durchlaufen.

Beispiel: Anwendung Kontrollstrukturen

Ermittlung der Zahlen zwischen 1 und 35, die sowohl durch 3 als auch durch 2 teilbar sind:

```
public class Teiler {
  public static void main(String[] args) {
    for(int i = 1; i <= 35; i++) {
      if((i % 3 == 0) & (i % 2 == 0)) {
        System.out.println(i + " ist durch 3 und durch 2 teilbar! ");
      }
    }
  }
}
```