

Variablen

Variablen sind Speicher für Zahlen, Texte und sonstige Objekte. Jede Variable hat einen Namen und einen Typ.

Beispiele für Datentypen sind:

Datentyp	Verwendung
int	ganze Zahlen
float	Zahlen mit Nachkommastellen
String	Texte

Die Java-Anweisung "int x;" deklariert die Variable x, in der man eine ganze Zahl speichern kann.

```
int x;    //Deklaration der Variablen x mit Datentyp int
```

Um in einer Variablen einen Wert zu speichern verwendet man die Zuweisung, die mit = geschrieben wird:

```
x = 0;    //0 in x speichern
```

Die folgenden Anweisungen deklarieren die Variable "t" und speichern den Text "Hallo Welt!" hinein.

```
String t;  
t = "Hallo Welt!";
```

Deklaration und Zuweisung dürfen auch in einer Zeile zusammengefasst werden:

```
int x = 0;  
String t = "Hallo Welt!";
```

Mit Hilfe von Variablen und Zuweisungen lassen sich Berechnungen durchführen.

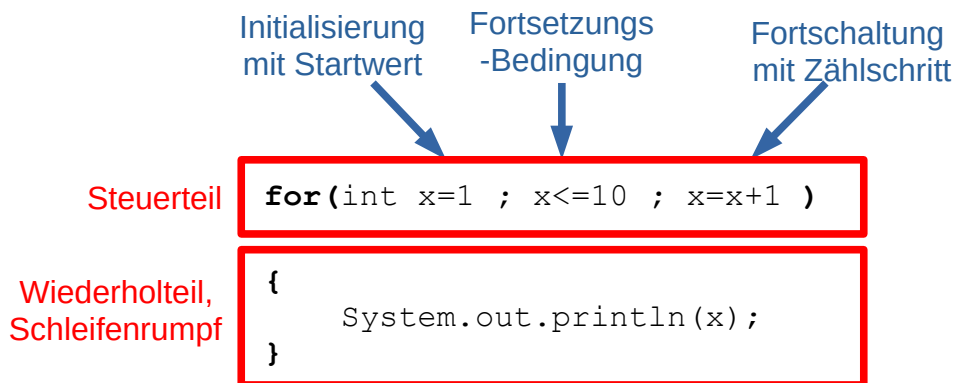
```
double a = 3.0 + 2.5;    //a erhält das Ergebnis 5.5  
int z = 5;               //z deklarieren und mit 5 initialisieren  
z = z + 1;               //Zum alten z-Wert (5) wird 1 addiert und das Ergebnis (6)  
                        //wieder in z gespeichert.
```

Zählschleifen (for)

Computer-Programme erledigen oft Aufgaben, die sich mehrmals wiederholen, wie z.B. Addieren mehrerer Zahlen oder Daten untereinander aufschreiben. In solchen Fällen wird häufig die Zähl- bzw. for-Schleife verwendet.

Das untenstehende Beispiel schreibt die Zahlen 1 bis 10 untereinander:

Bild 1. Teile der for-Schleife



Zählschleifen bestehen aus einem Steuerteil, der festlegt, wo mit der Zählung (1) begonnen und aufgehört (10) wird, welche Zählvariable (x) dabei zu verwenden ist und mit welcher Schrittweite (1) gezählt wird.

Wenn obige Schleife startet, dann wird zuerst x deklariert und auf 1 gesetzt (`int x=1`). Danach prüft der Computer ob die Fortsetzungsbedingung (`x<=10`) erfüllt ist. Wenn ja, dann führt er alle Anweisungen im Schleifenrumpf aus. Danach bearbeitet er die Fortschaltungsanweisung (`x=x+1`) und hinterher erneut die Fortsetzungsbedingung. Wenn sie immer noch erfüllt ist, dann werden der Schleifenrumpf und die Fortschaltungsanweisung erneut bearbeitet. Dies geschieht so lange, wie die Bedingung erfüllt ist.

Ist die Fortsetzungsbedingung nicht mehr erfüllt, dann endet die Schleife und es wird mit dem Befehl unter dem Schleifenrumpf fortgesetzt.

Beispiel: Alle geraden Zahlen von 10 bis 100 ausgeben:

```
for(z=10; z<=100; z=z+2) {  
    System.out.println(z);  
}
```

Arrays

Möchte man größere Datenmengen (z.B. mehrere Zahlen) bearbeiten, dann sind einfache Variablen ungeeignet, weil sie immer nur einen einzigen Wert enthalten können. Arrays dagegen bieten Platz für sehr viele Werte. Die haben wie Variablen einen Typ und einen Namen, zusätzlich aber auch eine Größe bzw. Länge.

```
int[] a = new int[5];    //Array mit Namen "a" und Platz für 5 int-Werte
```

Um gezielt auf einen einzelnen Speicherplatz innerhalb des Arrays zuzugreifen, erhält jeder Platz eine Nummer, den Index. Der erste Platz hat immer Index 0, alle weiteren Plätze werden einfach hochgezählt.

0	1	2	3	4

Den Index schießt man in eckige Klammer hinter dem Arraynamen ein:

```
y[0] = 12;  //Array mit Werten füllen  
y[1] = 7;  
y[2] = 17;  
y[3] = -2;  
y[4] = 0;
```

Das Ergebnis sieht dann so aus:

0	1	2	3	4
12	7	17	-2	0

Arrays können auch sofort bei der Erzeugung mit Werten gefüllt werden:

```
int[] x = { 24, 94, 200, 331, 469, 600, 706, 776};
```

Es ist erlaubt, den Index durch eine int-Variable oder einen Rechenausdruck zu ersetzen, der als Ergebnis einen int-Wert liefert. Dadurch ist es möglich, Arrays mit Hilfe von Zählschleifen zu bearbeiten.

Nachfolgendes Beispiel schreibt alle Inhalte des Arrays y auf den Bildschirm.

```
for(int i=0; i<y.length; i++) {  
    System.out.println(y[i];  
}
```

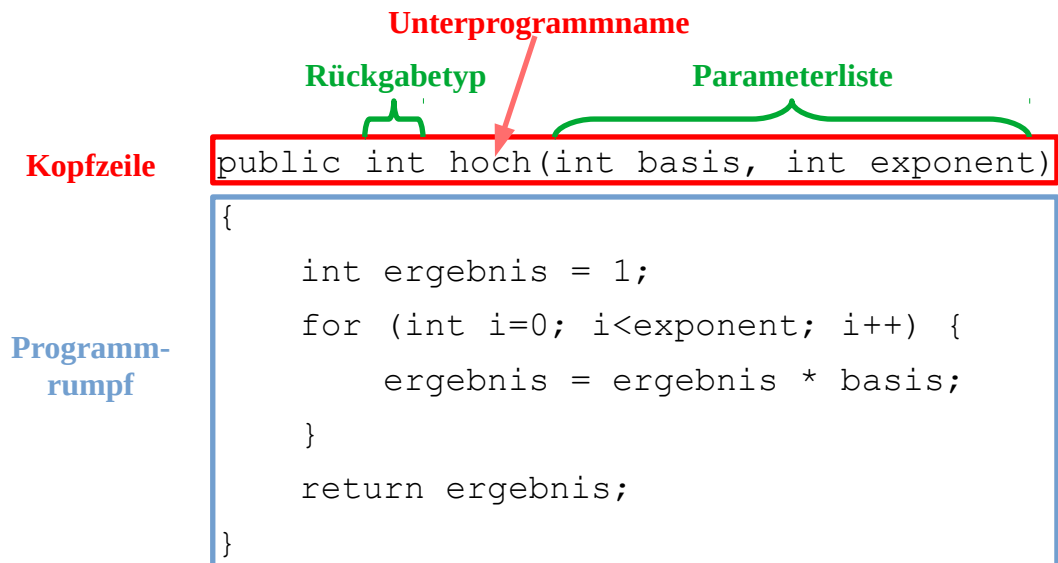
Das Konstrukt "y.length" liefert die Anzahl der Plätze, die Arrays y hat, also 5.

Unterprogramme / Methoden

Um in großen Programmen den Überblick zu behalten, zerlegt man sie in mehrere kleine Unterprogramme, von denen jedes nur eine spezielle Teilaufgabe erledigt.

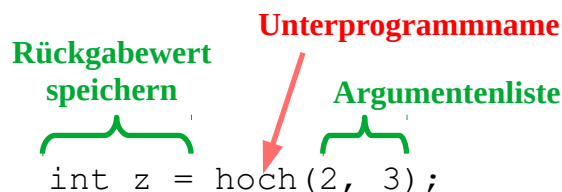
Ein Unterprogramm besteht aus einer Kopfzeile, den Name, die Parameterliste und den Rückgabebetyp enthält. Nach der Kopfzeile folgt in geschweiften Klammern der Unterprogrammumpf, der die Anweisungen enthält.

Bild 2. Unterprogrammdefinition.



Unterprogramme werden von anderen (Unter)programmen gestartet bzw. aufgerufen, in dem man den Namen gefolgt von der Argumentenliste angibt. Wenn das Unterprogramm Ergebnisse (=Rückgabewerte) liefert, dann sollten sie in einer Variablen gespeichert werden.

Bild 3. Unterprogrammaufruf.



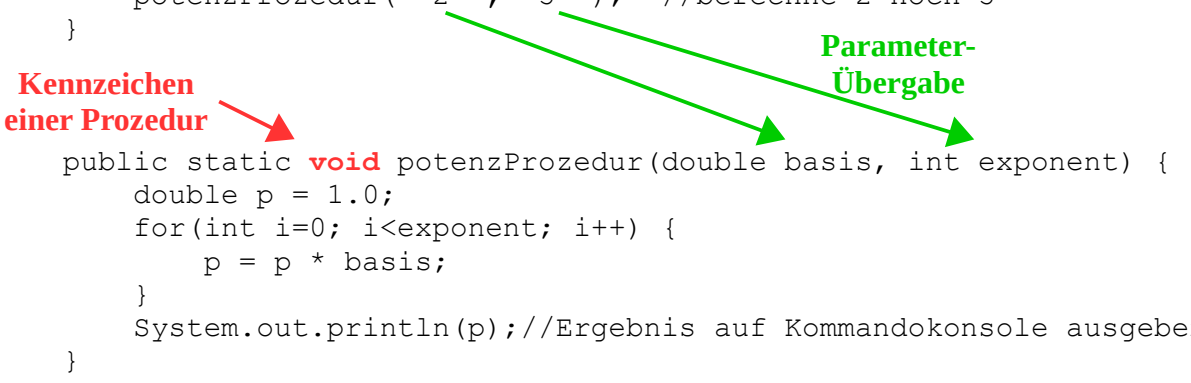
Es gibt zwei Arten von Unterprogrammen/Methoden:

- Prozeduren: ohne Rückgabewert
- Funktionen: mit Rückgabewert

Prozedur

Bild 4. Prozedur mit Aufruf.

```
class Potenz {  
    public static void main(String[] args) {  
        potenzProzedur( 2 , 5 ); //berechne 2 hoch 5  
    }  
  
    public static void potenzProzedur(double basis, int exponent) {  
        double p = 1.0;  
        for(int i=0; i<exponent; i++) {  
            p = p * basis;  
        }  
        System.out.println(p); //Ergebnis auf Kommandokonsole ausgeben  
    }  
}
```

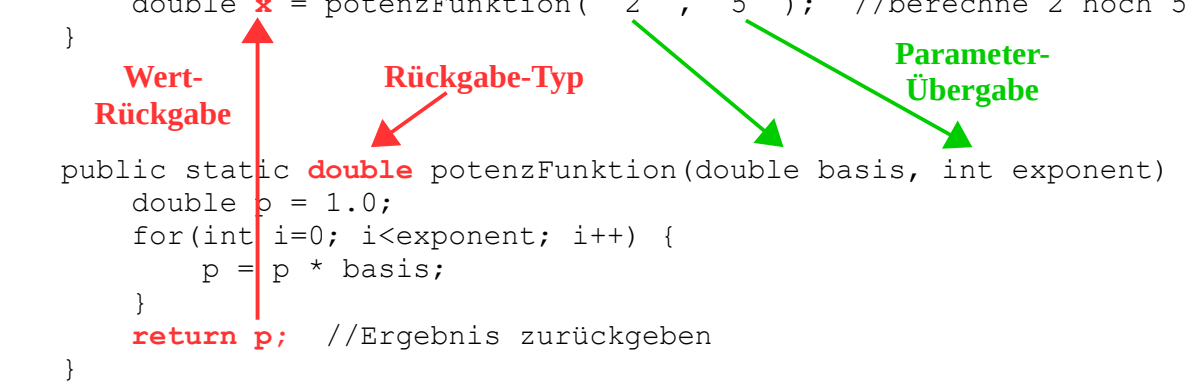


Prozeduren haben den Nachteil, dass Ergebnisse nicht mehr weiterverarbeitet werden können, da sie nur auf den Bildschirm ausgegeben werden. In diesem Fall ist es besser eine Funktion zu schreiben.

Funktion

Bild 5. Funktion mit Aufruf.

```
class Potenz {  
    public static void main(String[] args) {  
        double x = potenzFunktion( 2 , 5 ); //berechne 2 hoch 5  
    }  
  
    public static double potenzFunktion(double basis, int exponent) {  
        double p = 1.0;  
        for(int i=0; i<exponent; i++) {  
            p = p * basis;  
        }  
        return p; //Ergebnis zurückgeben  
    }  
}
```



Funktionen haben den Vorteil, dass ihre Rückgabewerte/Ergebnisse in einer Variablen gespeichert (x) und weiterverwendet werden können.

CSV-Dateien

CSV (=comma separated values) ist ein weit verbreitetes Dateiformat für tabellenartige Daten. Dabei werden in jeder Zeile die Spaltenwerte nebeneinander mit einem Komma dazwischen abgespeichert.

Bild 6. csv-Datei

```
name,cpu,id,path
explorer,80.66,14308,C:\WINDOWS\Explorer.EXE
FreeCommander,139.25,8196,C:\Portable\FreeCommander\FreeCommander.exe
notepad++,1.03,5936,C:\Portable\npp.7.1.bin\notepad++.exe
OneDrive,43.88,2384,C:\Users\Andi\AppData\Local\Microsoft\OneDrive\OneDrive.exe
OneDrive,39.98,8944,C:\Users\Andi\AppData\Local\Microsoft\OneDrive\OneDrive.exe
ONENOTEM,0.09,6488,C:\Program Files\Microsoft Office\root\Office16\ONENOTEM.EXE
PowerMgr,2.11,1220,C:\WINDOWS\SysWOW64\Lenovo\PowerMgr\PowerMgr.exe
powershell,43.47,8968,C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
RuntimeBroker,1.28,14736,C:\Windows\System32\RuntimeBroker.exe
SettingSyncHost,0.16,12908,C:\WINDOWS\system32\SettingSyncHost.exe
sihost,6.03,5976,C:\WINDOWS\system32\sihost.exe
soffice,0.03,4848,C:\Program Files\LibreOffice\program\soffice.exe
soffice.bin,64.89,9620,C:\Program Files\LibreOffice\program\soffice.bin
svchost,3.64,3636,C:\WINDOWS\system32\svchost.exe
```

Obiges Beispiel zeigt eine csv-Datei mit 4 Spalten, einer Kopfzeile und 14 Datenzeilen. Die Werte einer Zeile sind mit Komma voneinander getrennt, so dass man sie den Tabellenspalten zuordnen kann. Die Tabelle würde dann aussehen wie unten abgebildet.

Bild 7. csv-Datei als Tabelle

name	cpu	id	path
explorer	80,66	14308	C:\WINDOWS\Explorer.EXE
FreeCommander	139,25	8196	C:\Portable\FreeCommander\FreeCommander.exe
notepad++	1,03	5936	C:\Portable\npp.7.1.bin\notepad++.exe
OneDrive	43,88	2384	C:\Users\Andi\AppData\Local\Microsoft\OneDrive\OneDrive.exe
OneDrive	39,98	8944	C:\Users\Andi\AppData\Local\Microsoft\OneDrive\OneDrive.exe
ONENOTEM	0,09	6488	C:\Program Files\Microsoft Office\root\Office16\ONENOTEM.EXE
PowerMgr	2,11	1220	C:\WINDOWS\SysWOW64\Lenovo\PowerMgr\PowerMgr.exe
powershell	43,47	8968	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
RuntimeBroker	1,28	14736	C:\Windows\System32\RuntimeBroker.exe
SettingSyncHost	0,16	12908	C:\WINDOWS\system32\SettingSyncHost.exe
sihost	6,03	5976	C:\WINDOWS\system32\sihost.exe
soffice	0,03	4848	C:\Program Files\LibreOffice\program\soffice.exe
soffice.bin	64,89	9620	C:\Program Files\LibreOffice\program\soffice.bin
svchost	3,64	3636	C:\WINDOWS\system32\svchost.exe

Da das CSV-Format ist nicht standardisiert, gibt es viele verschiedene Varianten. Manchmal fehlt die Kopfzeile, als Trennzeichen werden oft auch Strichpunkte oder Tabulatorzeichen verwendet. In manchen Dateien sind die Werte in Anführungszeichen oder einfache Hochkomma eingeschlossen.

Einlesen einer csv-Datei

Um eine csv-Datei auszuwerten, empfiehlt es sich, zuerst alle Daten einzulesen und in einem oder mehreren Arrays abzuspeichern. Dort kann man dann nach bestimmten Einträgen suchen, Berechnungen durchführen usw.

Das Einlesen geschieht in mehreren Schritten, wie untenstehendes Bild zeigt. Die Schritte 1 bis 5 müssen solange wiederholt werden, bis alle Zeilen bearbeitet sind.

Bild 8. csv-Daten in Arrays einlesen.

process.csv

```
explorer,80.66,14308,C:\WINDOWS\Exp  
FreeCommander,139.25,8196,C:\Portab  
notepad++,1.03,5936,C:\Portable\npp  
OneDrive,43.88,2384,C:\Users\Andi\A  
OneDrive,39.98,8944,C:\Users\Andi\A  
ONENOTEM,0.09,6488,C:\Program Files  
PowerMgr,2.11,1220,C:\WINDOWS\SysWO  
powershell,43.47,8968,C:\Windows\Sy  
RuntimeBroker,1.28,14736,C:\Windows  
SettingSyncHost,0.16,12908,C:\WINDO  
soffice,0.03,4848,C:\Program Files\  
soffice.bin,64.89,9620,C:\Program F  
svchost,3.64,3636,C:\WINDOWS\system
```

1.) Zeile lesen

```
explorer,80.66,14308,C:\WINDOWS\Explorer.EXE
```

2.) in Felder zerlegen

explorer

80.66

14308

C:\WINDOWS\Explorer.EXE

3.) speichern

4.) in double
umwandeln

5.) in int
umwandeln

name[]

0	explorer
1	..
2	
3	
4	

cpu[]

0	80.66
1	..
2	
3	
4	

id[]

0	14308
1	..
2	
3	
4	

Untenstehende Tabelle zeigt das entsprechende Java-Programm.

Tabelle 1. Java-Programm.

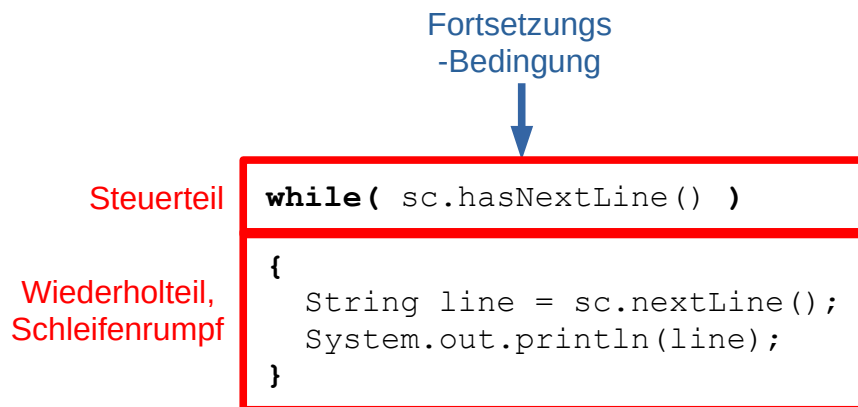
<code>String[] name = new String[100];</code>	Array für 100 Namen
<code>double[] cpu = new double[100];</code>	Array für 100 double-Werte
<code>int[] id = new int[100];</code>	Array für 100 int-Werte
<code>public void importCsv(String fileName) throws Exception {</code>	Prozedur, die im Fehlerfall (z.B. Datei nicht gefunden) mit einer Exception endet.
<code>FileInputStream fin; fin = new FileInputStream(fileName);</code>	Datei zum Lesen öffnen. Mit <code>FileInputStreams</code> kann nur byteweise gelesen werden.
<code>Scanner sc = new Scanner(fin);</code>	Scanner erzeugen, der auf <code>FileInputStream</code> zugreift.
<code>int z=0;</code>	Zähler für gelesene Zeilen
<code>while(sc.hasNextLine()) {</code>	Prüfen, ob noch ungelesene Zeilen vorhanden. Falls ja, dann Schleife fortsetzen, sonst beenden.
<code>String line = sc.nextLine();</code>	Eine Zeile aus Datei lesen.
<code>String[] fields = line.split(",");</code>	Zeile anhand der Kommata in mehrere Felder zerlegen.
<code>name[z] = fields[0];</code>	Erstes Feld in name-Array an Position z speichern.
<code>cpu[z] = Double.parseDouble(fields[1]);</code>	Zweites Feld in double konvertieren und in cpu-Array an Position z speichern.
<code>id[z] = Integer.parseInt(fields[2]);</code>	Drittes Feld in int konvertieren und in id-Array an Position z speichern.
<code>z++;</code>	Zeilenzähler erhöhen
<code>}//while</code>	
<code>sc.close();</code>	Scanner, <code>FileInputStream</code> und Datei schließen.
<code>}</code>	

Kopfgesteuerte Schleifen (while)

Viel Probleme mit sich wiederholenden Arbeitsschritten lassen sich mit Zähl- bzw. for-Schleifen lösen. Allerdings muss dabei die Anzahl der erforderlichen Wiederholungen im Voraus bekannt sein.

Ist die Wiederholungsanzahl nicht bekannt, wie z.B. beim Einlesen von Dateien, dann sind kopfgesteuerte bzw. while-Schleifen oft besser geeignet. Diese Schleifen haben im Steuerteil nur eine Bedingung, die entscheidet, ob der Wiederholteil ein weiteres mal ausgeführt werden soll.

Bild 9. Teile einer while-Schleife.



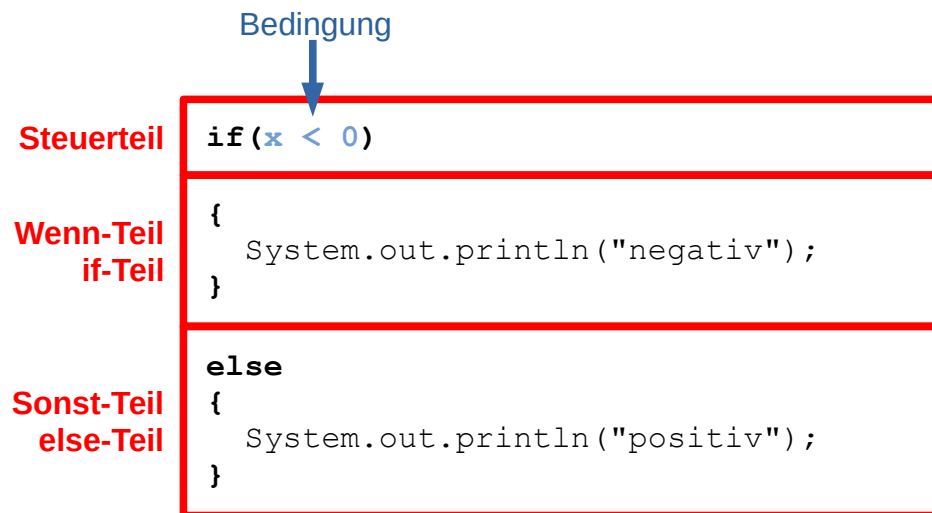
Beispiel: So lange Würfeln, bis 6 herauskommt.

```
int z = 0;  
while(z != 6) {  
    System.out.println(z);           //Testen, ob z ungleich 7  
    z = (int)(Math.random() * 6 + 1); //z ausgeben  
}
```

Verzweigungen (if-else)

Oft muss ein Programm Entscheiden, welche Befehle als nächstes ausgeführt werden. Um solch eine Entscheidung treffen zu können, muss eine Bedingung vorhanden sein, wie z.B. ist Variable x kleiner als 0? In untenstehendem Beispiel wird das Wort „negativ“ ausgegeben, wenn Bedingung $x < 0$ erfüllt ist, sonst wird „positiv“ ausgegeben.

Bild 10. Teile einer if-else-Verzweigung.

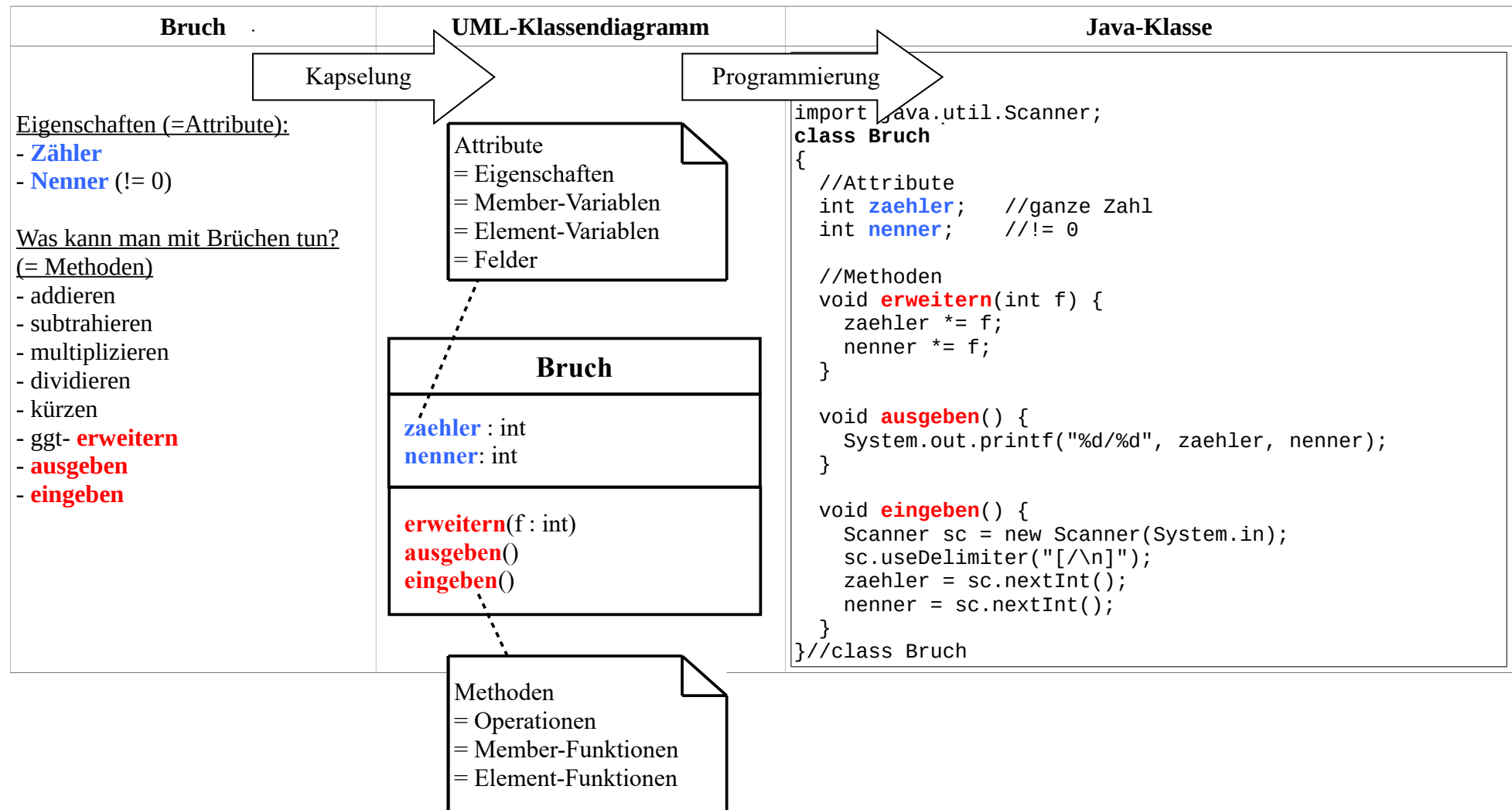


Je nach Anwendungsfall, darf der Sonst- bzw. else-Teil auch entfallen. Möchte man z.B. von einer beliebigen Zahl x den Betrag ermitteln (positiver Wert), dann genügt es, die Zahl mit -1 zu multiplizieren, wenn sie kleiner als 0 ist.

Beispiel: Betrag von Variable x bilden.

```
if(x < 0) {           //Testen, ob x negativ
    x = -1 * x;       //Minus-Vorzeichen von x entfernen
}
```

Die Klasse Bruch



Von der Bruch-Klasse zum Bruch-Objekt

Bruch.java

```
class BruchTest
{
    public static void main(String[] args) {
        Bruch b = new Bruch();    Objekterzeugung,
                                Instanzierung
        b.eingeben();//b einlesen
        ba.ausgeben();
    }
}
```

Klassendefinition

Begriffe

Eine Klasse

- ist ein Behälter für Attribute und Methoden
- ist ein Bauplan für Objekte (besteht aus Attributen und Methoden)
- ist ein Typ (für Variable bzw. Objekte)
- ist eine Menge gleichartiger Objekte (z.B. Menge aller Kraftfahrzeuge)

Ein Objekt

- ist ein Exemplar bzw. eine Instanz mindestens einer Klasse
- repräsentiert einen Gegenstand

Zugriffsschutz für Attribute und Methoden

Der Zugriff auf Attribute und Methoden kann eingeschränkt werden.

Schutz	Bezeichnung	UML-Symbol	Java-Schlüsselwort	Zugriff für
kein	öffentlich	+	public	jedes Programm erlaubt
sehr hoch	privat	-	private	für eigene Klasse

Wird der Zugriffsschutz verletzt, dann gibt der Compiler eine Fehlermeldung aus.

Die Vererbung / Generalisierung

Für ein Zeichenprogramm sollen zunächst die verschiedenen Arten von grafischen Bestandteilen festgelegt werden.

Bestandteile grafischer Darstellungen:

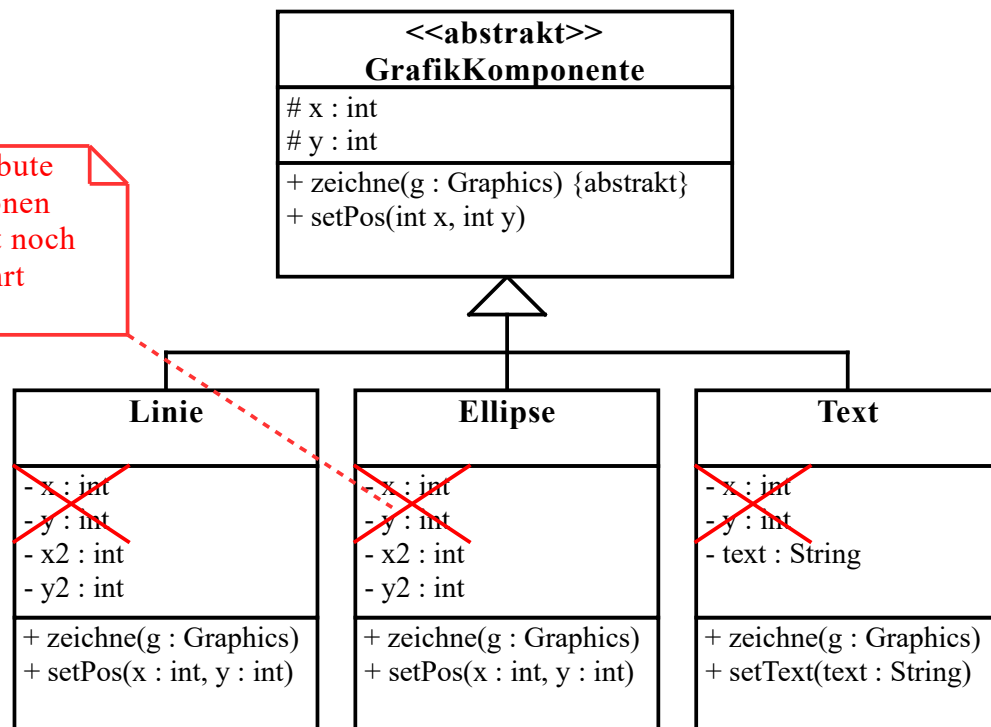
Bestandteil	Operationen/Methoden	Attribute
Linien	zeichne, setPos, ...	x, y, x2, y2
Ellipsen	zeichne, setPos, ...	x, y, x2, y2
Texte	zeichne, setPos, setText, ...	x, y, text

Klassendiagramm:

Basis-Klasse
= Super-Klasse
= Ober-Klasse

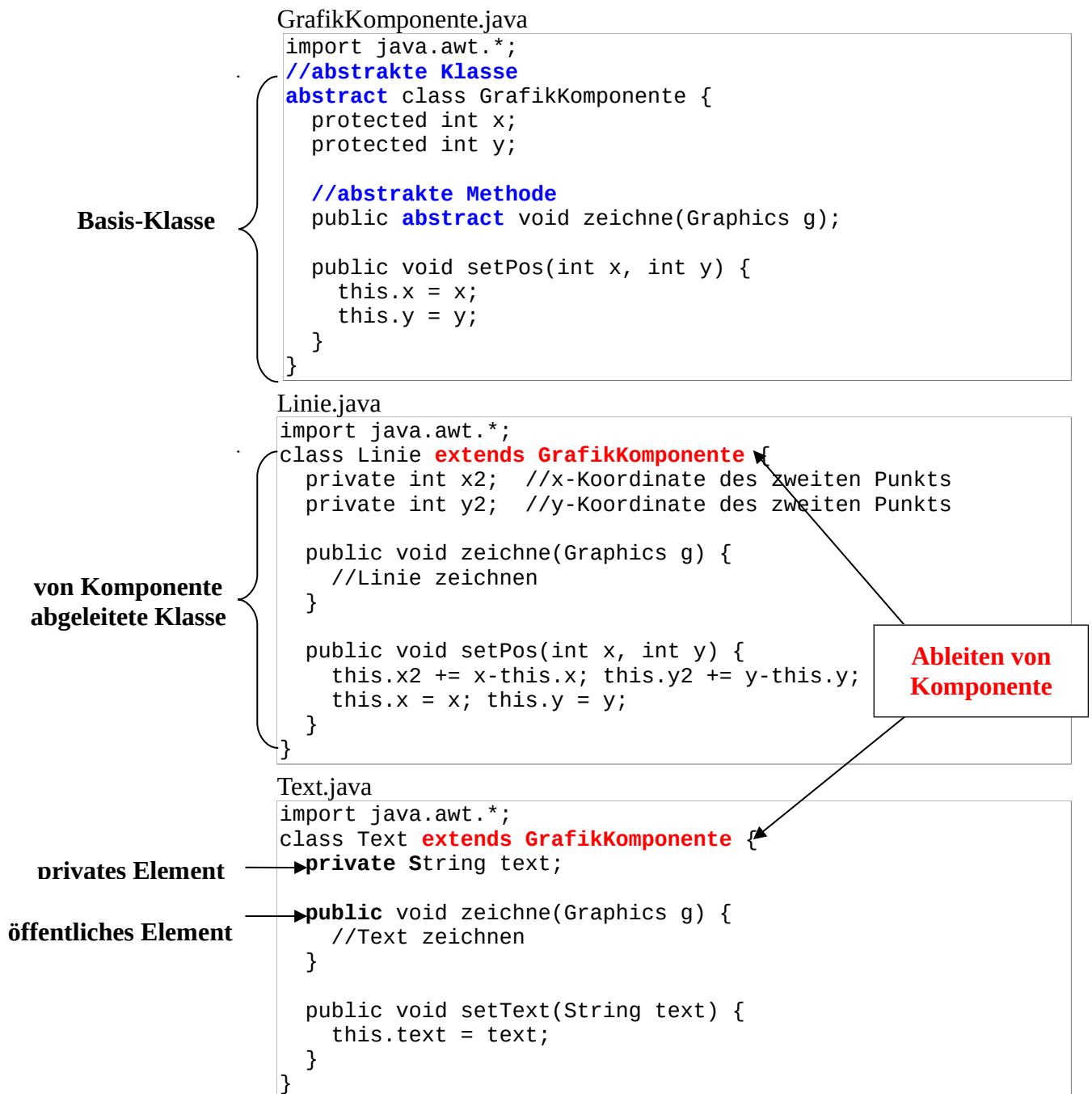
geerbte Attribute
und Operationen
müssen nicht noch
mal aufgeführt
werden!

Abgeleitete Klassen
= Sub-Klassen
= Unter-Klassen



- Die Attribute „x“ und „y“ werden von der Oberklasse übernommen bzw. geerbt und daher in den Unterklassen nicht mehr eingetragen.
- Die abstrakte Methode „zeichne“ erhält erst in den Unterklassen einen Programmrumpf, sie muss daher in den Unterklassen angegeben werden.
- Die Methode „setPos“ bekommt bereits in der Basisklasse einen Rumpf, der x und y auf neue Werte setzt. Dieser Rumpf wird auf alle Unterklassen vererbt.
- Da die Klassen Linie und Ellipse über ein zweites Koordinatenpaar (x2; y2) verfügen, muss setPos() in diesen Klassen mit einem neuen Rumpf überschrieben werden, so dass das Objekt verschoben wird, ohne seine Form zu verändern (Parallel-Verschiebung).

Java-Programm (vereinfacht)



- Sobald in einer Klasse abstrakte Methoden enthalten sind, ist die ganze Klasse abstrakt.
- Von abstrakten Klassen können keine Objekte erzeugt werden.