WAS IST EIN PROGRAMM

Ein Programm ist eine vom Computer ausführbare Datei. Es beinhaltet generell sogenannten Maschinencode. Dieser wird zur Ausführung in den Arbeitsspeicher des Rechners geladen. Dort wird das Programm als Abfolge von Maschinen-/Prozessorbefehlen von den Prozessoren des Computers verarbeitet und ausgeführt. Der Maschinencode ist Prozessorspezifisch.

Unter Computerprogramm wird auch der Quelltext des Programms verstanden, aus dem im Verlauf der Softwareentwicklung der ausführbare Code entsteht. Der Quelltext wird von den Programmierend erstellt und ist typischerweise in einer höheren Programmiersprache geschrieben. Höhere Programmiersprachen sind problemorientierte Sprachen die eine abstrakte und für den Menschen leichter verständliche Ausdrucksweise erlauben. Der Quelltext daraus kann automatisiert über einen Compiler oder Interpreter in Maschinensprache übersetzt werden.

Maschinencode (hexadezimal)
55 48 89 E5
C7 45 FC 02
C7 45 F8 03
8B 45 F8 8B 55 FC 01 D0 89 45 F4
8B 45 F4
5D C3

Beispiel Maschinencode (Wikipedia)

JAVA

Java ist eine objektorientierte (werden wir im Verlauf dieses Kurses sehen was das heisst) höhere Programmiersprache. Eine Eigenschaft von Java ist, dass der Java-Compiler den Quellcode in Java-Bytecode umwandelt / übersetzt. Java-Bytecode ist Maschinencode der von einer Virtuellen Maschine verstanden wird. Von der sogenannten JVM (Java Virtual Maschine). Zweck dieser Virtualisierung ist Plattformunabhängigkeit. Das Programm läuft so auf jeder Hardware auf der die passende Laufzeitumgebung also die JVM installiert ist.

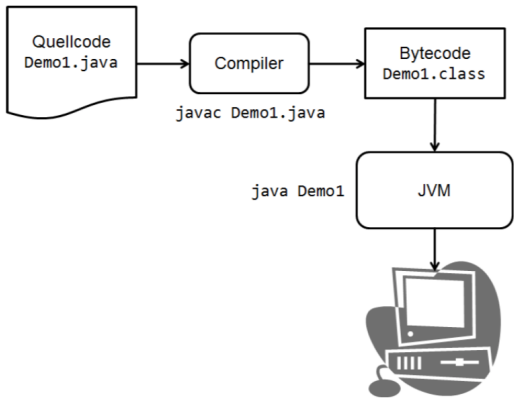


Abbildung 1-2: Übersetzung und Ausführung

Schematische Darstellung (Grundkurs Java)

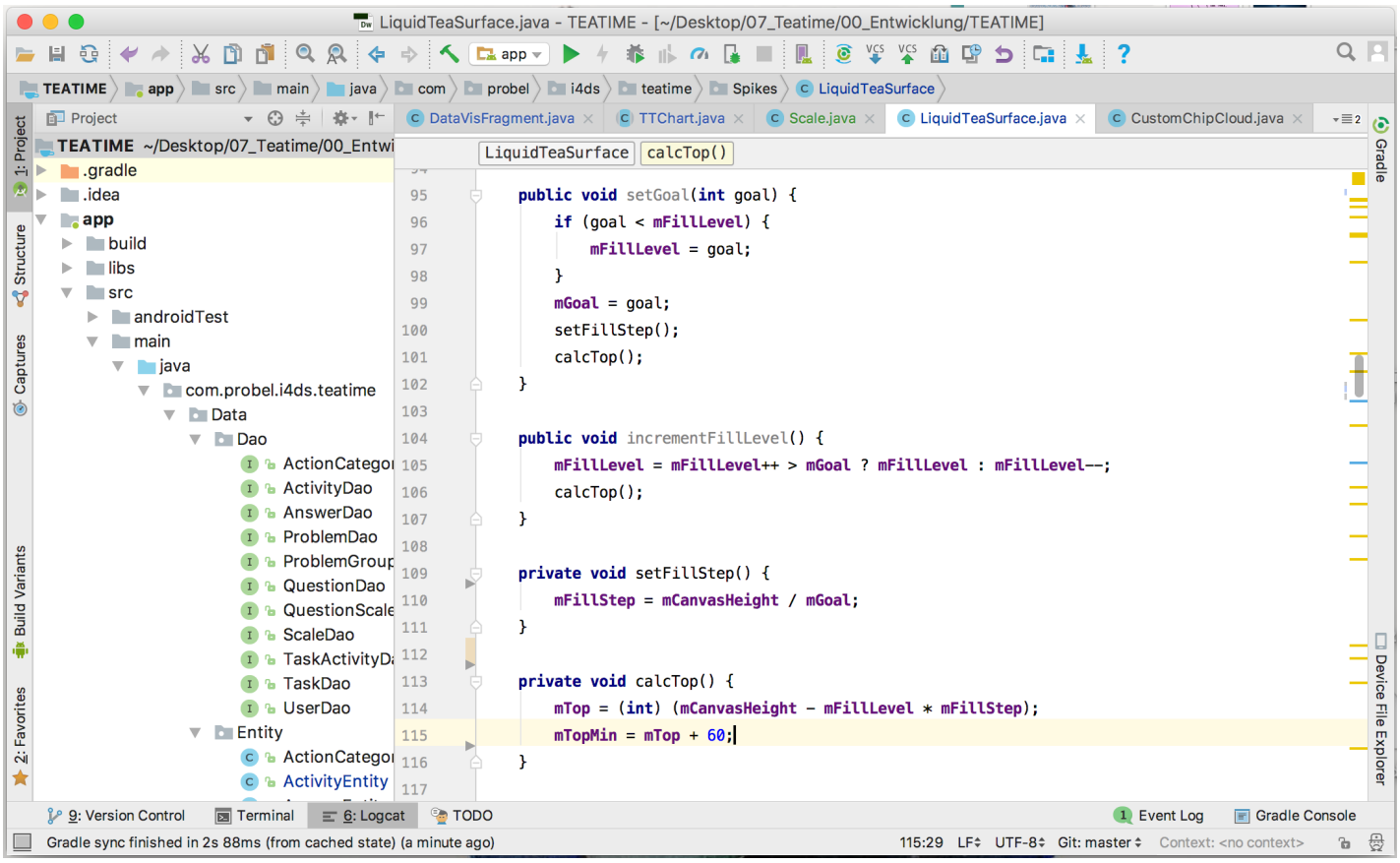
Consider the following Java code:

```
outer:
for (int i = 2; i < 1000; i++) {
    for (int j = 2; j < i; j++) {
        if (i % j == 0)
            continue outer;
    }
    System.out.println (i);
}
```

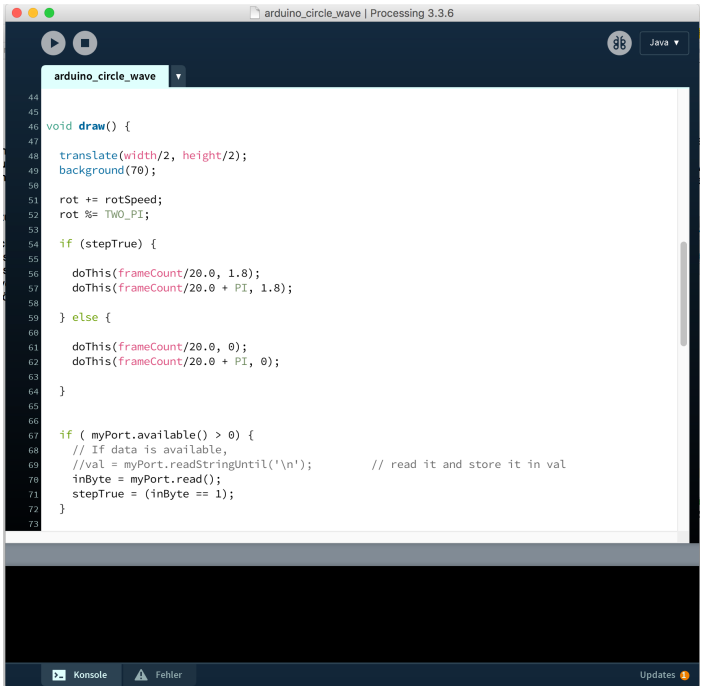
A Java compiler might translate the Java code above into byte code as follows, assuming the above was put in a method:

```
0:   iconst_2
1:   istore_1
2:   iload_1
3:   sipush  1000
6:   if_icmpge       44
9:   iconst_2
10:  istore_2
11:  iload_2
12:  iload_1
13:  if_icmpge       31
16:  iload_1
17:  iload_2
18:  irem
19:  ifne    25
22:  goto    38
25:  iinc    2, 1
28:  goto    11
31:  getstatic       #84; // Field java/lang/System.out:Ljava/io/PrintStream;
34:  iload_1
35:  invokevirtual   #85; // Method java/io/PrintStream.println:(I)V
38:  iinc    1, 1
41:  goto    2
44:  return
```

Um Programme in höheren Programmiersprachen zu entwicklen empfiehlt es sich eine IDE (integrierte Entwicklungsumgebung) zu verwenden. IDEs sind Software welche Softwareentwicklung ohne Medienbrüche ermöglichen und den Entwickelnden den Alltag erleichtern in dem sie hilfreiche Funktionalitäten, wie Syntax-Highlighting, das Kompilieren durch einen Klick oder das vereinfachte Einbinden von Libraries zur verfügung stellen.



Android Studio - IDE zur Android App Entwicklung



Processing IDE

VARIABLEN UND DATENTYPEN

Eines der Grundelemente der Programmierung sind die Variablen. Einer Variablen entspricht ein Speicherplatz im Arbeitsspeicher und sie ermöglicht es uns so Daten zu speichern. In Java wird jeweils definiert welchen Datentyp eine Variable besitzt. Die Variable kann dann nur Werte dieses Typs aufnehmen. Java kennt acht so genannte primitive Datentypen. Die primitiven Datentypen können Zahlen, Zeichen und Wahrheitswerte charakterisieren. Ein primitiver Datentyp ist charakterisiert durch einen festen Wertebereich.

Eine Variable wird mit folgender Syntax Deklariert und Initialisiert:

```
        frei gewählter Name dieser Variable
int name = 1;
Datentyp        Wert
```

Eine Variable muss nicht von Anfang an Initialisiert (ersten Wert zuweisen) werden sie kann auch nur deklariert (Typ und Namen festlegen) werden. Nachdem sie deklariert wurde könne wir die Variable im Programm beliebig oft verwenden und auch ändern.

```
int variableName;       Deklaration: Typ und Namen festlegen
…
variableName = 1;       Initialisierung: initial/erstmalig mit Wert belegen
```

Wahrheitswerte		
boolean	false, true	`boolean b = true;`

Ganze Zahlen		
byte	-128 - 127	`byte b = 1;`
short	-32’768 - 32’767	`short s = 1;`
int	-2’147’483’648 - 2’147’483’647	`int i = 1;`
long	-9.223.372.036.854.775.808 - 9.223.372.036.854.775.807	`long l = 1L;`

Gleitkommazahl		
float	ca $1{,}4 * 10^{-45}$ - $3{,}4 * 10^{38}$ Genauigkeit ca 7 Stellen	`float f = 1.1f;`
double	ca. $4{,}9 * 10^{-324}$ - $1{,}8 * 10^{308}$ Genauigkeit ca. 15 Stellen	`double d = 1.1;`

Processing benutz im Gegensatz zu reinem Java den Typ `float` als Standard Typ für eine Gleitkommazahl. Hier muss ein `double` mit `d` gekennzeichnet werden `double = 1.1d;`

Zeichen		
char	Unicode Zeichen	`char c = 'a';`
String	Zeichenketten	`String s = "Hallo Welt!";`

Der Typ `String` gehört nicht zu den primitiven Datentypen, wird hier aber trotzdem aufgeführt.

Zusatz - Wertebereich

Jeder der acht Datentypen hat eine bestimmte Anzahl Bits (1 Bit kann nur den Wert 0 oder 1 annehmen) zur Verfügung im Speicher. Woraus der fixe Wertebereich resultiert. Die Daten werden binär gespeichert.

Der Datentyp `int` zum Beispiel hat eine Grösse von 32 Bit, was in einen Wertebereich von -2'147'483'648 - 2'147'483'647 ergibt.
Beim `int` steht das erste Bit für das Vorzeichen. So ergeben 32 Einsen den Minimalwert -2'147'483'648 und eine Null und 31 Einsen den Maximalwert: 2'147'483'647.

Zusatz - default Initialisierung

Primitive Datentypen haben jeweils eine default Initialisierung. Schon bei der Deklaration wird der Variablen also einen default Wert zugewiesen, wenn wir sie nicht selber Initialisieren. Die default Werte für die primitiven Datentypen sind wie folgt:

Zahlen	0
Gleitkommazahlen	0.0
Wahrheitswert	false
Zeichen	‚\u0000'

für Referenzypten:

String	null

Auch wenn der default Wert dem Wert entspricht mit dem man die Variable gerne Initialisieren möchte, sollte man den Wert trotzdem selber Initialisieren um die Lesbarkeit des Codes zu gewährleisten.

OPERATOREN

Mit Operatoren können Zuweisungen und Berechnungen vorgenommen und Bedingungen formuliert und geprüft werden. Es gibt Operatoren für Berechnungen, zum Vergleichen von numerischen Werten und zum verknüpfen von Logischen Werten.

Operatoren für Berechungen

+	`int c = a + b;`	Addiert die Werte von `a` und `b` und speichert das Resultat in `c`.
-	`int c = a - b;`	Subtrahiert die Werte von `a` und `b` und speichert das Resultat in `c`.
*	`int c = a * b;`	Multipliziert die Werte von `a` und `b` und speichert das Resultat in `c`.
/	`int c = a / b;`	Dividiert die Werte von `a` und `b` und speichert das Resultat in `c`.
%	`int c = a % b;`	Berechnet den Modulo von `a` und `b` und speichert das Resultat in `c`.
++	`a++;`	Erhöht `a` um eins.
- -	`b--;`	Verkleinert `b` um eins.

Zusatz - Verkürzte Schreibweise

Die Operationen für Addition, Subtraktion, Multiplikation und Division können auch verkürzt geschrieben werden:

```
b += a;
```

Hier wird `a` zu `b` addiert und das Resultat in `b` gespeichert.

Zusatz - Präfix- und Postixform von ++ und - -

Die Operatoren ++ und - - wird die Präfix- und die Postfixform unterschieden.

`++a` hat den Wert von a+1, a wird um 1 erhöht
`--a` hat den Wert von a-1, a wird um 1 verringert
`a++` hat den Wert von a, a wird um 1 erhöht,
`a--` hat den Wert von a, a wird um 1 verringert.

Zusatz - Berechnungen mit unterschiedlichen Typen

Werden bei Berechnungen **zwei Werte** von u**nterschiedlichem Typ** verrechnet so **wird der „kleinere“** der beiden **automatisch** in **den „grösseren“** umgewandelt. Verrechnet man also eine Variable vom Typ `int` mit einer vom Typ `double`, werden beide zu `double`. Das Resultat hat den Typ des grösseren Operanden, ist aber mindestens vom Typ `int`.

Zusatz - String Konkatenation

Verschiedene `Strings` können durch **+** zusammengesetzt werden.

```
String hello = "Hello" + "World!";
```

(Diese Operation ist eher aufwendig und sie werden im weiteren Verlauf ihres Studiums eine bessere Methode kennen lernen.)

Operatoren zum Vergleichen		
<	`a < b`	kleiner
<=	`a <= b`	kleiner gleich
>	`a > b`	grösser
>=	`a >= b`	grösser gleich
==	`a == b`	gleich
!=	`a != b`	ungleich

Logische Operatoren (Verknüpfung von Wahrheitswerten)	
!	nicht
&	und (vollständig)
^	xor
|	oder (vollständig)
&&	und (kurz)
||	oder (kurz)

Logische Operatoren Verknüpfen Wahrheitswerte miteinander. Für die Operatoren UND und ODER gibt es eine kurze und eine vollständige Auswertung. Bei der vollständigen Auswertung werden alle Operanden berechnet, während bei der kurzen der zweite Operand nicht mehr ausgewertet wird sobald das Ergebnis der Gesamtwertes fest steht.

a	**b**	**a & b** **a && b**	**a | b** **a || b**	**a ^ b**
true	true	true	true	false
true	false	false	true	true
false	true	false	true	true
false	false	false	false	false

ARRAYS

Soll eine Sammlung von Elementen desselben Datentyps gespeichert werden kann dies mit einem Array erreicht werden. Arrays sind keine primitiven Datentypen sondern Referenztypen. Das heisst sie haben keinen festgelegten Wertebereich. Es wird im Hintergrund eine Referenz zum Speicherplatz an dem die Sammlung abgelegt ist gespeichert. Für uns wird diese Referenz über den Namen repräsentiert.[1]

Ein Array wird mit folgender Syntax Erzeugt:

```
        frei wählbarer Name          Anzahl Elemente, die die Sammlung beinhaltet (ganzzahlig)
int[] arrayName = new int[10];
Datentyp                 Ein Referenztyp wird über die Anweisung new Erzeugt
```

Ein Array kann auch als Variable deklariert und später im Code Erzeugt werden:

```
int[] arrayName;
…
arrayName = new int[10];
```

Oder er wird direkt bei der Deklaration Initialisiert. (Erstmals Daten zugewiesen)

```
int[] x = {1, 10, 4, 0};
          Array x ist eine Sammlung von 4 Elementen
```

Um auf ein bestimmtes Element eines Arrays zugreifen zu können wird der Name des Arrays und die Position (Index) auf die man Zugreifen möchte angegeben. Jedes Element kann wie eine normale Variable verwendet werden.

```
                                    Name        Position
int elementAnPositionZwei = arrayName[2];
```

hier wird der Element Wert der im Array gespeichert ist einer neuen Variable zugewiesen

```
arrayName[2] = 3;
```

hier wird dem Element an Position 2 den Wert 3 zugewiesen

MERKE:
Die Elemente im Array werden von 0 an gezählt. Ein Array mit Länge 3 kann also auf folgende Elemente zugreifen: array[0], array[1] und array[2]!

Möchte man auf die Anzahl der Elemente in einem Array zugreifen kann man das über das length Attribut machen.

```
int arraySize = arrayName.length;
```

[1] https://processing.org/reference/Array.html
https://processing.org/reference/arrayaccess.html

KONTROLLSTRUKTUREN

BLOCK { … }

Die geschweiften Klammern { und } fassen mehrere Anweisungen zu einem Block zusammen. Blöcke können auch ineinander geschachtelt werden. **Variablen, die in einem Block definiert werden, sind nur dort gültig und sichtbar.** Die Variablen in einem Block müssen einzigartig benannt werden.

```
{
  int variable = 1;
  String variable = "hello";
}
```

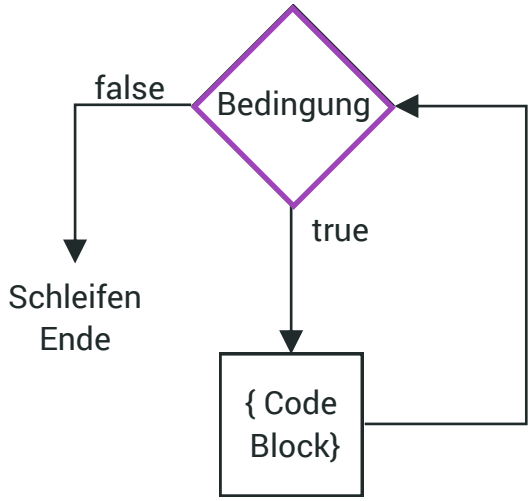
Duplicate local variable variable

SCHLEIFEN

Schleifen führen einen Block von Anweisungen wiederholt durch, solange eine Bedingung erfüllt ist. Eine Bedingung ist ein Boole'scher Ausdruck (Beisp: für `int x = 1; int y = 2;` hat der Boole'sche Audruck **`x < y`** den Wert `true`. Die Bedingung wäre erfüllt.)

In Java gibt es zwei Arten von Schleifen. Die while- und die for-Schleife.

Die while-Schleife
Die while-Schleife wird durch das Statement `while` definiert, welchem die Bedingung folgt. Darunter kommt der Code Block der solange wiederholt wird wie die Bedingung gültig ist.




```
int max = 10;
int sum = 10;

while ( sum < max ){
  sum = sum + 2;
}
println(sum);
```

Die for-Schleife
Bei der for-Schleife kann neben der Abbruchsbedingung auch einen Zählern/Laufvariable definiert werden. Diese verändert sich nach jedem Durchlauf. So wird nach dem `for` Statement die Zählervariable initialisiert, dann die Bedingung definiert und zum Schluss festgelegt, wie sich die Zählervariable nach jedem Schleifendurchlauf verändert.

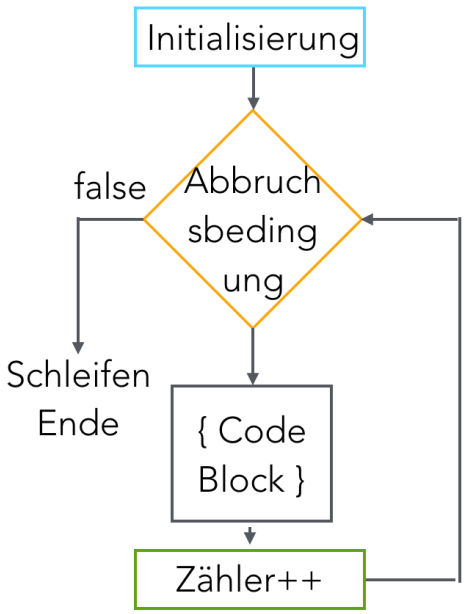
```
int max = 10;


for( int i = 0; i < max; i++){
  // Anweisungs Block
}
```

Update der Zähler Variable

Abbruchsbedingung

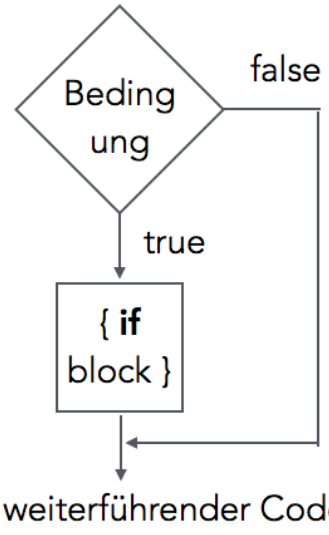
Initialisierung der Zähler Variable vom Typ int



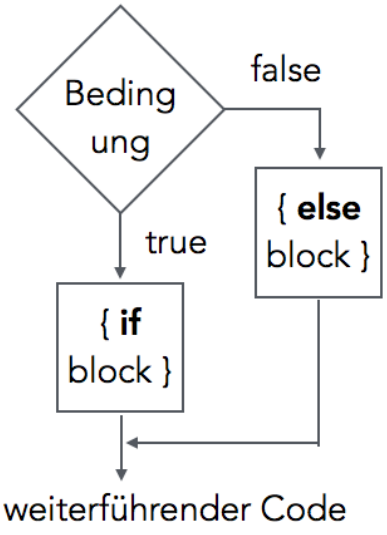
IF / ELSE VERZWEIGUNG

Verzweigungen ermöglichen es einen Code Block nur bedingt auszuführen. Die `if`-Anweisung tritt in zwei Varianten auf:

```
if( a < b ){
  // if code block
}
```




```
 if( a < b ){
  // if code block
}else{
  // else code block
}
```



METHODEN

Eine Methode ist dazu da bestimmte Code-Partien auszugliedern und wiederverwendbar zu machen. So kann man eine bestimmte Funktionalität, wie etwa das Berechnen der Y-Koordinate auf einer Linie für eine gegebene X-Koordinate, programmieren und ihr einen (Methoden)namen zuweisen mit dem man genau diese Funktionalität immer wieder verwenden kann. Das hat den Vorteil, dass man erstens weniger Code schreiben muss, man zweitens nur an einer Stelle etwas ändern muss, wenn die Funktionalität angepasst werden soll und man drittens den Code leserlicher macht (wenn man selbsterklärende Methodennamen verwendet).

Methoden können sich (in etwa) wie mathematische Funktionen vorgestellt werden. Eine Methode definiert, wie ein allfälliger Input Wert prozessiert wird und kann einen Wert zurück gegeben.
Die Methodensyntax sieht wie folgt aus:

```
Rückgabetyp Name( //optional Methodenparameter ){
      //Methoden Code
}
```

Wenn wir also zum Beispiel eine Geradenfunktion programmieren möchten sähe das wie folgt aus:
Geradenfunktion : f(x) = 2 * x + 0 beziehungsweise y = 2* x + 0
Wir können x als Inputwert und y als Rückgabewert sehen. Wenn wir nämlich zum Beispiel x = 9 setzen erhalten wir für y: 2*9+0 = 18
als Methode können wir das wie folgt umsetzen:

```
int line(int x){
    int y = 2*x+0;
    return y;
}
```

die Methode kann wie folgt an einem anderen Ort im Code aufgerufen werden:

```
line(9);
```

Der Methodenaufruf führt dazu, dass der Methodencode ausgeführt wird und gibt dann den Returnwert zurück. Der Methodenaufruf nimmt den Rückgabewert an.

Im folgenden Processing Beispielcode wurde die Methode `line()` definiert. Sie wird aufgerufen und es wird 2 übergeben. Nachdem der Methodenblock ausgeführt wurde wird der Rückgabewert in der `result` Variable gespeichert und dann auf der Konsole ausgegeben.

```
1
2  void setup(){
3    int result = line(2);
4    println("Das Resultat ist: ",result);
5  }
6
7  int line(int x){
8    int y = 2 * x + 1;
9    return y;
10 }
11
```

```
Das Resultat ist:  5
```

Rückgabewerte

`return`
Um einen Wert zurück zu geben benutzen wir das `return` Statement. Das return-Statement gibt den Wert der folgt zurück und schliesst die Methode ab. (Jeglicher Code der nach einem return kommt, wird nicht mehr ausgeführt.)

`void`
Soll die Methode keinen Rückgabewert haben müssen wir das definieren indem wir den Rückgabewert mit `void` angegeben. Hier braucht es natürlich auch kein return-Statement.

Methodenparameter

```
int methodName(int x, String word, double size){   //3 Methodenparameter
      //x, word und size sind im Methodenblock gültige Variablen
}
```

Methodenparameter definieren welchen Input die Methode entgegennimmt. Wenn keine angegeben sind, nimmt die Methode keine entgegen. Methodenparameter werden wie Variablen mit Typ und Name angegeben und können im Methodenblock auch wie normale Variablen verwendet werden.

KLASSEN UND OBJEKTE