



## 2.6.2 Objektorientierte Programmierung

### Arbeitsblatt 01 Einführung in Processing

#### Einführung in Processing

Für die Programmierung in Processing steht eine integrierte Entwicklungsumgebung (**IDE** – engl. integrated development environment) zur Verfügung. Eine IDE ermöglicht das Erstellen von Computerprogrammen in einer Programmieroberfläche.

1. Gib in Processing die Anweisungen so ein, wie sie in der Abbildung unten dargestellt sind.

Lege die Verzeichnisstruktur <> Processing > Programme > siebzehnundvier an.

- Speichere das Programm unter der Bezeichnung `siebzehnplusvier`.

Processing legt für jedes Programm ein eigenes Verzeichnis an: `sketch_01_siebzehnplusvier`

In diesem Verzeichnis ist die Programmdatei `sketch_01_siebzehnplusvier.pde`. (pde steht für processing development environment).

Die Bezeichnung `sketch` (engl. für „Skizze“) soll darauf hindeuten, dass man in Processing schnell Zeichnungen erzeugen kann.

Das Programm wird in dem **Texteditor** eingegeben.

- Starte das Programm, indem du auf die Schaltfläche klickst.

In der **Konsole** können Textausgaben angezeigt werden.

Für Grafikausgaben wird nach dem Programmstart ein **Zeichenfenster** geöffnet.

The screenshot shows the Processing IDE interface. The top bar displays the title "sketch\_01\_siebzehnplusvier | Processing 3.3.3" and menu options like Datei, Bearbeiten, Sketch, Debugger, Tools, and Hilfe. Below the menu is a toolbar with a play button, a stop button, and other icons. The main area is a text editor containing the following code:

```
1 println("Das ist Processing:");
2 print("17+4=");
3 println(17+4);
4 println("17+4=" + 17+4);
5 print("17+4=" + (17+4));
```

Below the text editor is a "Konsole" window showing the program's output:

```
Das ist Processing:
17+4=21
17+4=174
17+4=21
```

A large bracket on the left side of the text editor and the console window groups the two together, indicating they are part of the same process. A smaller bracket groups the text editor and the play button, indicating they are related.

- Beende das Programm, indem du auf die Schaltfläche klickst.
- Vergleiche die Ausgaben in der Konsole mit den Anweisungen des Programms. Beschreibe die Wirkung der Anweisungen in den einzelnen Programmzeilen.

`print:` \_\_\_\_\_

`println:` \_\_\_\_\_

1: \_\_\_\_\_

2: \_\_\_\_\_

3: \_\_\_\_\_

4: \_\_\_\_\_

5: \_\_\_\_\_



## 2.6.2 Objektorientierte Programmierung

Arbeitsblatt 01 Einführung in Processing

## Variablen

2. Wie lautet die Anweisung für die Zeile 6? Ergänze und teste das Programm.  
Speichere den Sketch als 02\_siebzehnplusvier.  
(vgl. ..\262-materialien\rechnung\sketch\_02\_siebzehnplusvier\_variable)

```
sketch_02_rechnung_variable
1 String Titel;
2 int Summe;
3 Titel="Processing mit Variablen:";
4 Summe=17+4;
5 println(Titel);
6
```

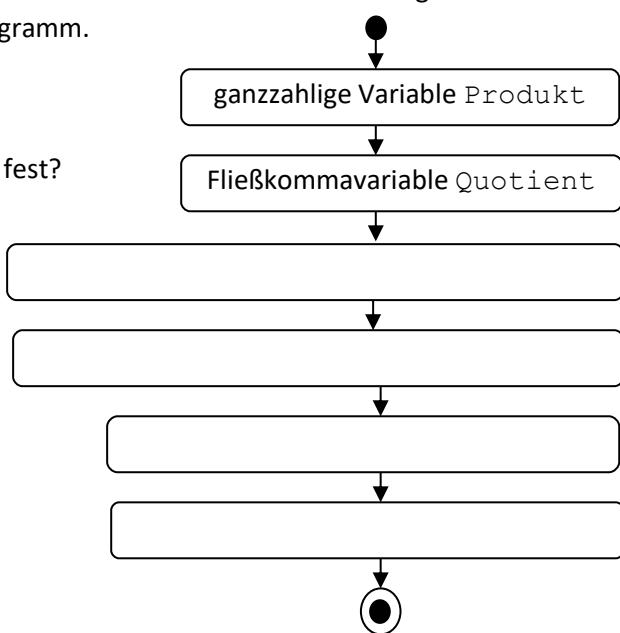
3. Implementiere ein Programm, das zwei Zahlen multipliziert und dann das Produkt durch eine andere Zahl dividiert.

Die Terme sollen in der Konsole angezeigt werden.

Achte darauf, dass der Quotient keine ganze Zahl ist.

Hinweis: Das Zeichen für Multiplikation ist \*, für Division /.

- Ergänze den Algorithmus in dem Aktivitätsdiagramm.
  - Codiere den Algorithmus in Processing und speichere den Sketch als 03\_rechnung.
  - Sieh dir das Ergebnis genau an. Was stellst du fest?



Das Problem kann man mit einer **Funktion** behoben werden. Das ist eine Art von Unterprogramm, das man in dem eigentlichen Programm aufrufen kann. Eine Programmiersprache verfügt über viele Funktionen.

Ein Beispiel ist die Funktion `float()`, die ganzzahlige Werte in Kommazahlen umwandelt.

**Statt Quotient=Produkt/5; kann man kann z. B. schreiben Quotient=float(Produkt)/5;**

Dadurch wird der Wert der Variablen `Produkt` vor der Division in eine Kommazahl umgewandelt.

4. Ergänze dein Programm (04\_rechnung\_funktion).



## 2.6.2 Objektorientierte Programmierung

### Arbeitsblatt 01 Einführung in Processing

#### Funktionen

5. Gib in Processing das Programm so ein, wie es in den Lerninhalten S. 3 beschrieben ist.  
Speichere das Programm unter der Bezeichnung `05_siebzehnplusvier_funktion`.

Hinweise:

The screenshot shows the Processing IDE interface. The code editor window has a title bar 'sketch\_05\_siebzehnplusvier\_funktion'. The code itself is:

```
1 void setup() {  
2     addiere();  
3 }  
4  
5 void addiere () {  
6     int Summe;  
7 }
```

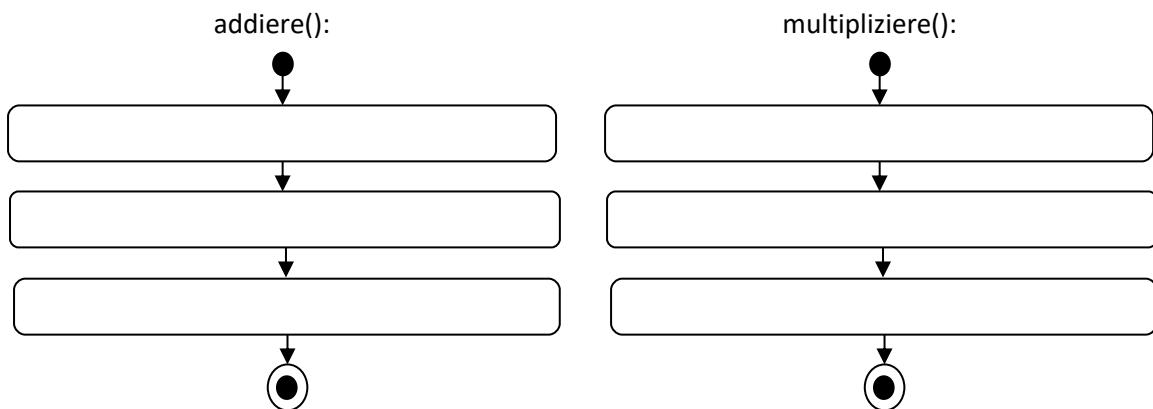
A red error bar at the bottom of the code editor highlights the line 'int Summe;' with the message 'Missing a semicolon ";"'. The status bar at the bottom shows the error message 'Missing a semicolon ";" sketch\_05\_siebzehnplusvier\_funktion 6'. Below the code editor are two tabs: 'Konsole' (which is selected) and 'Fehler'.

Während der Eingabe erhältst du in der Statuszeile Warn- und Fehlerhinweise

Mit den Schaltflächen unten kannst du zwischen der Konsole und den manchmal etwas ausführlicheren Fehlermeldungen hin- und herschalten.

6. In einer weiteren Funktion sollen zwei Zahlen multipliziert werden.

- o Ergänze die Aktivitätsdiagramme für die Funktionen `addiere()` und `multipliziere()`.



- o Codiere den Algorithmus (`06_rechnung_funktion`).

7. Zusatzaufgabe: In einer weiteren Funktion soll der Quotient aus 17 und 4 berechnet und in der Konsole angezeigt werden (`07_division`).



## 2.6.2 Objektorientierte Programmierung

### Arbeitsblatt 01 Einführung in Processing

#### Klassen

In EOS stehen eine ganze Reihe fertiger Klassen zur Verfügung, z. B. Kreis oder Rechteck.  
Bei objektorientierter Programmierung kann man auch eigene Klassen erstellen, was in EOS nicht geht.  
Für die Schreibweise in Processing wird der zusammenhängende Programmcode dargestellt:

Beispiel:

```
Addierer Addierer1; // Objekt der Klasse Addierer mit dem Bezeichner "Addierer1".  
  
void setup() {  
    Addierer1=new Addierer(); // Ein Objekt der Klasse benötigt einen Objektbezeichner und  
    Addierer1.gibAus(); // wird mit der Anweisung new erzeugt.  
}  
  
class Addierer { // Dabei wird der Konstruktor aufgerufen.  
    int Summe; // Danach kann eine Methode des Objekts aufgerufen werden.  
    Addierer() {  
        Summe=17+4; // Eine Klasse wird mit dem Schlüsselwort class begonnen.  
    } // Damit wird ein neuer Datentyp festgelegt, der wiederum Variablen und  
    void gibAus () { // Funktionen beinhalten kann.  
        println("17+4="+Summe); // Die Variablen innerhalb einer Klasse werden als Attribute bezeichnet.  
    } // In dem Konstruktor werden die Attribute mit Default-Werten belegt.  
}
```

8. Codiere die Klasse Addierer (08\_siebzehnplusvier\_klasse).

9. Ergänze eine Klasse Multiplizierer (09\_rechnung\_klasse).  
Gib in dem Klassendiagramm rechts das Attribut und die Methode an.

Klassendiagramm:

Multiplizierer

#### Attribute und Punktnotation

10. Ändere das Programm 09\_rechnung\_klasse so ab, dass in der Klasse Multiplizierer die zuvor berechnete Summe mit einer Zahl malgenommen wird.

Du kannst auch die Vorlagendatei sketch\_v01\_rechnung\_klasse verwenden.  
Speichere den Sketch unter 10\_rechnung\_Attribut.

- Beschreibe die Meldung in der Statuszeile.

- 
- 
- Wird eine Variable innerhalb einer Klasse deklariert, handelt es sich um ein **Attribut**.  
Auf ein Attribut kann aber nur in Bezug auf ein Objekt zugegriffen werden.

11. Teste den Zugriff auf das Attribut mit Hilfe der **Punktnotation** Addierer1.Summe.

(11\_rechnung\_Punktnotation)

- Mit Hilfe der Punktnotation Objekt.Attribut bzw. Objekt.Methode kann auf ein Attribut bzw. auf eine Methode in Bezug auf ein Objekt zugegriffen werden.

12. Zusatzaufgabe: Ergänze die Klasse Dividierer, in der das Produkt durch die Summe dividiert wird (12\_division; siehe Klassendiagramm rechts).

Dividierer
Quotient:int
gibAus ()