

Der Werkzeugkasten

Alle Klassen der Bibliothek *Stifte und Mäuse* auf einen Blick

Behalte dieses Heft das ganze Halbjahr

Dieses Heft sagt dir, **welche Werkzeuge es gibt** — nicht, was du damit bauen sollst. Genau das ist der Punkt: Zwei Programme, die dieselben Werkzeuge benutzen, können völlig verschieden aussehen. In jedem Projekt entscheidest du selbst, welche Werkzeuge du brauchst.

Jedes Programm beginnt so:

```
import sum.kern.*; // Bildschirm, Stift, Buntstift, Maus, Tastatur, Farbe
import sum.werkzeuge.*; // Uhr, Rechner

public class MeinProgramm
{
    Bildschirm derBildschirm; // Objekte, die das Programm benutzt
    Buntstift meinStift;

    public MeinProgramm() // Konstruktor: hier werden die Objekte erzeugt
    {
        derBildschirm = new Bildschirm(1000, 700);
        meinStift = new Buntstift();
    }

    public void fuehreAus() // dein eigentliches Programm
    {
        // ...
    }
}
```

Bildschirm — die Zeichenfläche

Ursprung (0,0) liegt **oben links**.

Die **v**-Achse zeigt nach **unten**.

Bildschirm()	Fläche in Maximalgröße
Bildschirm(b,h)	Fläche mit Breite und Höhe
breite()	liefert die Breite
hoehe()	liefert die Höhe
loescheAlles()	Fläche wird leer
setzeFarbe(f)	Hintergrundfarbe (löscht!)
holeGanzeZahl(t)	fragt eine ganze Zahl ab
holeZahl(t)	fragt eine Kommazahl ab
holeText(t)	fragt einen Text ab
nachVorn()	Fenster in den Vordergrund

Stift — zeichnet Spuren

Der Stift hat **Position**, **Richtung** (0° = nach rechts, gegen den Uhrzeigersinn), **Zustand** (oben/unten) und **Modus**.

runter() / hoch()	Spur an / aus
bewegeUm(d)	d Pixel in aktueller Richtung
bewegeBis(h,v)	direkt zu einem Punkt
dreheUm(w)	um w Grad weiterdrehen
dreheBis(w)	auf w Grad einstellen
dreheZu(h,v)	zu einem Punkt hindrehen
zeichneKreis(r)	Kreis um die Position
zeichneRechteck(b,h)	Position = linke obere Ecke
schreibeText(s)	schreibt <i>über</i> der Position
schreibeZahl(z)	schreibt eine Zahl
hPosition()	liefert die h-Koordinate
vPosition()	liefert die v-Koordinate
winkel()	liefert die Richtung
istUnten()	true, wenn der Stift zeichnet
normal()	zeichnet in Stiftfarbe
radriere()	zeichnet in Hintergrundfarbe
wechsle()	tauscht Farbe und Untergrund

Buntstift — ein Stift, der mehr kann

Der **Buntstift** erbt alles vom **Stift** und kann zusätzlich:

<code>setzeFarbe(f)</code>	z. B. <code>Farbe.ROT</code>
<code>setzeLinienbreite(b)</code>	dickere Linien
<code>setzeFuellmuster(m)</code>	<code>Muster.GEFUELLT</code> u. a.
<code>setzeSchriftgroesse(g)</code>	Textgröße
<code>setzeSchriftart(a)</code>	<code>Schrift.ARIAL</code> u. a.
<code>setzeSchriftstil(s)</code>	<code>Schrift.FETT</code> u. a.
<code>textbreite(s)</code>	wie breit wird der Text?

Farben: `Farbe.SCHWARZ`, `BLAU`, `CYAN`, `DUNKELGRAU`, `GRAU`, `GRUEN`, `HELLGRAU`, `MAGENTA`, `ORANGE`, `PINK`, `ROT`, `WEISS`, `GELB` — oder selbst gemischt: `Farbe.rgb(r,g,b)` mit Werten von 0 bis 255

Muster: `Muster.DURCHSICHTIG`, `DURCHSCHEINEND`, `GEFUELLT`

Tastatur

<code>wurdeGedrueckt()</code>	liegt ein Zeichen bereit?
<code>zeichen()</code>	liefert das Zeichen (char)
<code>weiter()</code>	Zeichen aus dem Puffer werfen

Immer in dieser Reihenfolge: erst `wurdeGedrueckt()` prüfen, dann `zeichen()` lesen, dann `weiter()` aufrufen. Sonst gibt es eine Fehlermeldung.

Sondertasten: `Zeichen.PFEILLINKS`, `PFEILRECHTS`, `PFEILOBEN`, `PFEILUNTEN`, `EINGABE`, `ESCAPE`, `TAB`, `F1...F12`

Uhr (aus `sum.werkzeuge`)

<code>warte(ms)</code>	Pause in Millisekunden
<code>starte() / stoppe()</code>	Stoppuhr
<code>gestoppteZeit()</code>	gestoppte Zeit in ms
<code>verstricheneZeit()</code>	Zeit seit dem Start
<code>zeit() / datum()</code>	als Text
<code>stunde() minute() sekunde()</code>	als Zahl

Maus

<code>istGedrueckt()</code>	true, solange gedrückt wird
<code>doppelKlick()</code>	true bei Doppelklick
<code>hPosition()</code>	h-Koordinate der Maus
<code>vPosition()</code>	v-Koordinate der Maus

*Es gibt **keine** Methode, die von allein auf einen Klick wartet. Das Warten musst du selbst programmieren — mit einer Schleife.*

Rechner (aus `sum.werkzeuge`)

<code>ganzeZufallsZahl(a,b)</code>	Zufallszahl von a bis b
<code>zufallsZahl()</code>	Kommazahl zwischen 0 und 1
<code>wurzel(z)</code>	Quadratwurzel
<code>quadrat(z)</code>	Quadrat
<code>sin(w) cos(w) tan(w)</code>	Winkelfunktionen
<code>gerundet(z)</code>	rundet auf eine ganze Zahl

Bausteine der Sprache

Die Formen, die du immer wieder brauchst

Nachschlagen erlaubt — auswendig lernen nicht nötig

Ein Objekt benutzen

```
meinStift.runter();
meinStift.bewegeUm(100);
double x = meinStift.hPosition();
```

Auftrag (macht etwas, gibt nichts zurück) gegen **Anfrage** (liefert einen Wert). Eine Anfrage steht rechts vom = oder in einer Bedingung.

Variablen

```
int versuche = 0;
double abstand = 0.0;
boolean getroffen = false;
String name = "Kreuztal";
char taste;
```

Bedingung

```
if (abstand < 30)
{
    // dann ...
}
else
{
    // sonst ...
}
```

Vergleiche: == != < > <= >=

Texte vergleicht man mit `s.equals("...")`

Logische Operatoren

&& UND — beides muss gelten
|| ODER — eines reicht
! NICHT — dreht die Bedingung um

```
if (h > 100 && h < 400)
if (!meineMaus.istGedrueckt())
```

Schleifen

vorprüfend — prüft, *bevor* etwas passiert:

```
while (!meineMaus.istGedrueckt())
{
    // warten ...
}
```

nachprüfend — läuft mindestens einmal:

```
do
{
    // ...
}
while (stift.vPosition() < 600);
```

Zählschleife — wenn die Anzahl feststeht:

```
for (int i = 0; i < 12; i = i + 1)
{
    // ...
}
```

Eigene Methode

```
private void zeichneRing(int r)
{
    meinStift.zeichneKreis(r);
}

private double abstand(int h, int v)
{
    return derRechner.wurzel(h*h + v*v);
}
```

Ohne Rückgabe: **void**. Mit Rückgabe: Datentyp davor-schreiben und **return** benutzen.

⚠ Fünf Fehler, die fast alle einmal machen

1. **Der Stift ist oben.** Es passiert scheinbar nichts — er bewegt sich, aber ohne Spur. **runter()** vergessen.
2. **Position (0,0).** Text wird *über* der Stiftposition geschrieben. Ganz oben links sieht man ihn deshalb nicht.
3. **Semikolon hinter while(...)** — die Schleife läuft dann leer und das Programm hängt.
4. **= statt ==** in einer Bedingung. Der Compiler meldet das meistens, aber nicht immer.
5. **v-Achse.** Größeres v bedeutet *weiter unten*. Wer nach oben will, muss *subtrahieren*.

📖 Nachschlagen im Buch *Informatik 1*

Das Buch arbeitet mit einer anderen Programmierungsumgebung (Greenfoot, Rover auf einem Planeten). Die **Konzepte** sind aber dieselben — lies dort und übersetze in die Sprache des Stifts.

S. 181	Fehlermeldungen des Compilers verstehen
S. 182	Kommentieren
S. 190-191	Struktogramme
S. 192	Stylekonventionen: wie sauberer Code aussieht
S. 193-194	Testen: wie man systematisch prüft